GXV-Ada Programmer's Guide

SunSoft
A Sun Microsystems, Inc. Business
2550 Garcia Avenue
Mountain View, CA 94043
U.S.A

Part No.: 802-3474-10
Revision A, November 1995

2550 Garcia Avenue, Mountain View, California 94043-1100 U.S.A.

Contents

Tables

Preface

GXV-Ada is a companion tool to the OpenWindows™ Developer's Guide (Devguide). It translates the Guide Interface Language (GIL) files, which were generated by the Devguide program, into Ada source code, utilizing the Ada XView bindings.

Who Should Read This Book

This manual is addressed to software developers who write and test Ada programs coded in text (ASCII) source.

Before You Begin

This manual assumes you are familiar with the OPEN LOOK® interface and the OpenWindows™ environment, particularly the use of the mouse to activate a window, select text, and click on buttons.

If you are not familiar with the OPEN LOOK interface, see the chapter on the OPEN LOOK GUI in *Managing SPARCworks Tools.* For more information on the OPEN LOOK GUI, see the *OPEN LOOK Application Style Guidelines.*

For more information on the OpenWindows environment, see *OpenWindows Developer's Guide 3.0.1 for X86 User's Guide.*

How This Book Is Organized

Chapter 1, "Introduction," provides an overview of Devguide and GXV-Ada. This chapter also lists the hardware and software requirements for using the tools.

Chapter 2, "Getting Started," describes how to run Devguide and how to use GXV-Ada to create interface source code. It then explains how to compile, link, and run the resultant output of GXV-Ada.

Chapter 3, "Generated Source Code," describes the files that GXV-Ada generates. This chapter is of particular interest to programmers who want to modify the generated source code and thoroughly link and integrate GXV-Ada source code with their own code.

Chapter 4, "Ada-Specific Information," is for programmers who need to know the differences that exist between GXV (C source code generation) and GXV-Ada. It provides details on how and why the resultant output differs.

This guide provides only basic information about Devguide required to use GXV-Ada. For more information about Devguide, see *OpenWindows Developer's Guide 3.0.1 User's Guide*.

*On-Line Release Notes (*README*)*

You can refer to the release notes (`README` file) in the following directory location:

```
/opt/SUNWspro/GAda3.0/README
```

This file contains information about known problems with the software.

What You Should Know Before Reading Further

This manual is intended for programmers who are familiar with OpenWindows, the OPEN LOOK user interface, XView, the Ada programming language, and SPARCompiler™ Ada. A working knowledge of the OpenWindows environment, including using the pointer and the SELECT, ADJUST, and MENU mouse buttons (or their equivalents) to work with windows, panes, menus, and controls, is assumed.

For more information about these topics, consult:

- *OPEN LOOK Graphical User Interface User's Guide* for information about the elements that constitute user interfaces in OpenWindows and the rules and suggestions for creating OPEN LOOK user interfaces
- *The Annotated Ada Reference Manual,* ANSI-MIL-STD-1815A-1983 (Annotated) 2nd Edition (or other suitable Ada language text) for the rules of programming in Ada
- *XView Programming Manual, Xlib Reference Manual, Xlib Programming Manual,* and *X Protocol Reference Manual* (published by O'Reilly and Associates, Inc.) for information about programming in the XView environment of OpenWindows
- *SPARCompiler Ada User's Guide, SPARCompiler Ada Programmer's Guide, SPARCompiler Ada Reference Guide, SPARCompiler Ada Runtime System Guide,* and *Multithreading Your Ada Applications,* for information about using SPARCompiler Ada
- *Installing SunSoft Developer Products on Solaris* for information about installing SPARCompiler Ada and GXV-Ada
- *OpenWindows Developer's Guide 3.0.1 Installation Manual* for information about installing Devguide.

What Typographic Changes and Symbols Mean

The following table describes the typographic conventions and symbols used in this book.

Table P-1 Notational Conventions

Typeface or Symbol	**Meaning**	**Example**
`AaBbCc123`	Names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `system% You have mail.`
`AaBbCc123`	What you type, contrasted with on-screen computer output	`system% su` `Password:`
AaBbCc123	Command-line placeholder: replace with a real name or value	To delete a file, type `rm` *filename*.

Table P-1 Notational Conventions *(Continued)*

Typeface or Symbol	Meaning	Example
AaBbCc123	Book titles, new words or terms, or words to be emphasized	Read Chapter 6 in *User's Guide*. These are called *class* options. You *must* be root to do this.
♦	A single-step procedure	♦ **Click on the Apply button.**
Code samples are included in boxes and may display the following:		
%	UNIX C shell prompt	`system%`
$	UNIX Bourne and Korn shell prompt	`system$`
#	Superuser prompt, all shells	`system#`

How to Get Help

SPARCworks tools include four on-line help facilities.

- **AnswerBook™ system —** an on-line documentation tool that displays this manual in full, along with the other SPARCworks tools manuals. You can read this manual on line and take advantage of dynamically linked headings and cross-references.

  To start the AnswerBook system, type: `answerbook`

- **On-line Help —** To access on-line help messages, place the pointer on the window, menu, or menu button, and then press the Help key.

- **Notices —** a standard feature of the OPEN LOOK environment. Notices serve two functions. Some notices are prompts that inquire about whether you want to continue with a particular action. Other notices are precautionary in that they provide information about the end result of a particular action. They appear only when the end result of an action is irreversible.

- **SunOS Manual Pages (man pages) —** on-line documentation about the command-line utilities of the SunOS operating system. To access the man page for FileMerge, type:

  ```
  man filemerge
  ```

Related Documentation

This manual is part of the SPARCworks Adadocument set. Other manuals in this set include:

- *Multithreading Your Ada Application*
- *SPARCompiler Ada Programmer's Guide*
- *SPARCompiler Ada Reference Guide*
- *SPARCompiler Ada Runtime System Guide*
- *SPARCompiler Ada User's Guide*
- *SPARCworks/Ada Tutorial*
- *SPARCworks/Ada User's Guide*
- *SPARCworks 3.0 AnswerBook*

These and other related documents in the on-line AnswerBook system. AnswerBook systems available for this release are:

- **Common Tools document**s **—** *SPARCworks/SPARCompiler 3.0 Related Material AnswerBook*
- **Installation Guide —** *SPARCworks/SPARCompiler 3.0 Installation AnswerBook*

To access the AnswerBook systems, refer to the installation manual.

Introduction 1

This guide describes how to use the Devguide GIL file translation tool: GXV-Ada. This tool generates Ada language source code from a Devguide Graphical Interface Language file (a GIL file).

Devguide is the OpenWindows Developer's Guide tool that you use to design prototype OpenWindows user interfaces. When you finish designing an interface in Devguide, you save the design in a GIL file. You then process the GIL file through a language- and GUI-toolkit-specific translation tool, such as GXV-Ada. The resulting Ada source code builds the actual user interface.

Devguide helps automate the design of OpenWindows user interfaces. You can also build an OpenWindows interface using calls to an XView library. GXV-Ada generates Ada source code that implements XView-based calls to user interface routines. Thus, if your end-product is to be an XView-based interface, you use GXV-Ada. When you use the tool, Ada source code is functionally the same, hence the actual user interfaces are the same.

In addition to generating Ada source code for calls to user interface routines, this translation tool also produces other files that help automate the task of building a user interface in Ada using Devguide.

GXV-Ada generates:

- A `Makefile`
- Package specification and body files
- "Stubs" files, which help you link the interface code to the main body of the application code.

1

1.1 More About Devguide

Devguide is a development tool designed to make the interface programmer's job easier. Using Devguide gives you the freedom to create and try user interfaces without ever writing a line of code.

Devguide operates in the OpenWindows application environment, a graphical user interface environment that operates on SPARC-based workstations. You use Devguide to create user interfaces that conform to the OPEN LOOK user interface as it is implemented in OpenWindows.

"Guide" in Devguide stands for Graphical User Interface Design Editor-05an accurate description of its function. You use Devguide to assemble the elements of a user interface by dragging visual representations of the elements onto the workspace of the monitor screen and putting the elements together to form a user interface for your application. When you finish assembling the elements of a user interface, you save the interface as an interface file (the GIL file), which you can recall later for modification or use to generate source code.

1.2 What You Need to Run GXV-Ada

GVX-Ada is a component of SPARCompiler Ada, which provides the Ada compiler, Ada XView, and Ada OLIT libraries necessary to compile a user interface developed in Devguide.

For information on installing SPARCompiler Ada, see the installation manual.

GVX-Ada also requires a system running the following software:

- OpenWindows version 3.3 or later

  OpenWindows provides the graphical user interface, window system, and toolkit necessary to run Devguide and the compiled GXV-Ada interface.

- Devguide, which is a tool for interactive development of a OpenWindows graphical user interface

  Devguide produces a graphic interface language (GIL) file, which GXV-Ada uses to generate SPARCompiler Ada user interface code by calls to Ada XView libraries. For information on installing Devguide, see *OpenWindows Developer's Guide 3.0.1 Installation Manual*

Getting Started 2

2.1 Running Devguide

To start Devguide, you must first run OpenWindows version 3. You can use File Manager to start Devguide by double-clicking SELECT on the Devguide glyph. You can also start Devguide from a Shell or Command Tool or shell script, as described below.

Before running Devguide, first set the appropriate variables-05`GUIDEHOME`, `PATH`, and `HELPPATH`. To do this, follow the procedure outlined in the *OpenWindows Developer's Guide 3.0.1 Installation Guide*.

To start Devguide from a Shell or Command Tool, type:

```
% devguide &
```

The Devguide base window then appears.

2.1.1 Choosing a Toolkit

Next, choose a toolkit to generate code for the user interface. After you choose Devguide from the base window Properties menu, you see the Properties window.

To use GXV-Ada:

- **Choose XView from the Toolkit menu (the default choice).**

Now you can use Devguide to construct a user interface. Refer to the *OpenWindows Developer's Guide 3.0.1 User's Guide* for information on building and testing a user interface.

2.1.2 Quitting Devguide

Before quitting Devguide, be sure to save your work.

To quit Devguide:

- **Choose Quit from the Devguide Window menu.**
  Devguide quits, along with any UI elements Devguide laid out on the workspace.

2.2 Generating Ada Source Code

After you have created and saved a user interface, you can use the companion toolkit code generator to generate Ada source code for the user interface.

To run GXV-Ada, enter the following command in an open Shell:

```
% gxv_ada filename
```

filename is the name of the GIL file. The GIL file extension, `.G`, is optional. GXV-Ada automatically adds the `.G` extension if you leave it off. For example, to generate GXV-Ada source code for the GIL file `display.G`, enter:

```
% gxv_ada display
```

The GXV-Ada tool creates several files in addition to the Ada source code files. These other files include a `Makefile` and an application help text file.

Devguide refers to a group of related files as project files. You can generate GXV-Ada source files for project files using the `-p` option to the tool command. As with the `.G` extension, the `.P` is optional. For example, enter either:

```
% gxv_ada -p myfile
```

or:

```
% gxv_ada -p myfile.P
```

Note that GXV-Ada must be run in the same directory as the `.G` or `.P` files; if not, provide a full path name to the tool or the code.

2.3 Compiling Object Code and Running the Results

To compile code generated with GXV-Ada, you must set the ADAHOME environment variable. ADAHOME designates the top-level directory of the SPARCompiler Ada release. The installer of SPARCompiler Ada should know the proper directory name, or you can examine the contents of the /etc/VADS file to determine the directory location.

2.3.1 ADAHOME *Environment Variable*

With the proper directory name of the SPARCompiler Ada release, you can set the ADAHOME environment variable:

If you use C shells, enter:

```
% setenv ADAHOME ada_location
```

If you use Bourne and Korn shells, enter:

```
% ADAHOME=ada_location
% export ADAHOME
```

Alternatively, place these commands in your home directory .profile or .cshrc file for automatic inclusion every time you create a shell.

SPARCompiler Ada generates dynamically linked executables. To successfully link and run GXV-Ada executables, the environment variable LD_LIBRARY_PATH must designate the proper library archive directories.

If you use Bourne and Korn shells, enter:

```
% LD_LIBRARY_PATH=$OPENWINHOME/lib:$GUIDEHOME/lib:\
  /usr/lib:/usr/ucblib
% export LD_LIBRARY_PATH
```

If you use C shells, enter:

```
% setenv LD_LIBRARY_PATH $OPENWINHOME/lib:$GUIDEHOME/lib:\
  /usr/lib:/usr/ucblib
```

The appropriate commands can be placed in your home directory .profile or .cshrc file. The $ADAHOME/xview/lib and libguidexv_iface/lib directories are required for GXV-Ada.

2.3.2 Makefile

The GXV-Ada code generator produces a `Makefile`. When you run `make`, the `Makefile` issues commands to create an Ada library, compile the source code, and link an executable.

To run `make`:

♦ **Enter** `make` **from a system prompt in a Command or Shell Tool.**

After successful completion of the `make` command, you can run the newly compiled user interface code by entering the name of the executable.

Generated Source Code 3

3.1 Preparing Devguide Interfaces for GXV-Ada

GIL files generated by Devguide can be sent through many different language-specific source code generators. Because each language has its own rules and restrictions, few code generators support all Devguide capabilities. The GXV-Ada code generator supports almost all Devguide capabilities. Few restrictions associated with Devguide-GIL files are generated through GXV-Ada.

3.1.1 Naming Conventions

As you create user interface elements in Devguide, you must assign a unique name to each element that is stored in a single GIL file. Unique names prevent problems with name-clashing during compilation (for example, multiply defined symbols). You can, however, use the same element name in separate GIL files.

Because windows tend to include parallel elements, developers are tempted to give those elements the same name. If you store each window in a separate GIL file, you can use the same name within each window without worrying about compilation errors.

However, the connections you specify for interface elements are global to an application and must have a unique name. You should call a connection by the same name every time you use it, whether it occurs in the same GIL file or across multiple GIL files.

Remember not to use an Ada reserved word for an interface element name. Also, avoid giving entities names of predefined Ada library units, types, constants, and subprograms. Follow the common Ada style of not hiding predefined language capabilities when choosing names for elements in Devguide. GXV-Ada relies on visibility to some predefined Ada types and library units. Hiding these names generally results in illegal compilations.

3.1.2 Devguide 1.1 GIL File Formats

GIL files created with Devguide version 1.1 might include unsupportable capabilities for GXV-Ada. Pass the Devguide 1.1 GIL files through Devguide version 3.0 and resave them to convert any version 1.1 capabilities to version 3.0 capabilities. The resulting version 3.0 GIL files can then be processed by GXV-Ada.

Note that some 1.1 files use predefined names of Ada library units, types, constants, and subprograms. You may need to modify these names.

3.2 Using GXV-Ada Source Code Files

After you create an interface and save it in one or more GIL files, you generate source code by executing the `gxv_ada` command, followed by either the name of one or more GIL files or by a project file. GIL files end in `.G` and project files end in `.P`. GXV-Ada generates three or more source code files, a help file, and a `Makefile` you can use with the `make` command to compile the source code files.

3.2.1 GXV-Ada Source Code File Names

GXV-Ada names its generated files after the original GIL file name, stripping off the `.G` extension and adding new extensions to identify each file. The four file name extensions are listed in Table 3-1.

Table 3-1 GXV-Ada Source Files

File Name	Description
`_ui.a`	Graphical user interface (GUI) data declarations
`_ui_b.a`	GUI code to implement the application
`_stubs.a`	Main program plus callback subprograms
`.info`	Help file

For example, if you use GXV-Ada to generate source code for a GIL file named `newt.G`, then the files `newt_ui.a`, `newt_ui_b.a`, `newt_stubs.a`, `newt.info`, and `Makefile` are generated.

3.2.2 *The* `_ui.a` *File*

The `_ui.a` file is a package specification containing declarations for each window instance structure. These declarations include types that contain handles to the window and each object within the window.

This file also contains Ada subprograms for each initialization and creation function in the `_ui_b.a` file. The instance initialization functions create a new copy of an instance and create each object in the instance by calling the object creation function.

Do not alter this file manually; any changes you make to this file will be written over by a subsequent invocation of GXV-Ada. Save custom declarations in your own files so they will not be lost during subsequent GXV-Ada invocations.

3.2.3 *The* `_ui_b.a` *File*

The `_ui_b.a` file is the primary source code file. It is the package body corresponding to the package specification defined in the `_ui.a` file. The `_ui_b.a` file creates the elements of the user interface. It begins with context clauses (Ada `with` and `use` clauses) and it follows immediately with one initialization subprogram per instance.

Do not alter the `_ui_b.a` file manually; any changes you make to this file will be written over by subsequent invocations of GXV-Ada and you will lose your modifications. If you want to make changes to objects in your interface, do so after the objects have been instantiated by function calls in the main program in the `_stubs.a` file.

3.2.3.1 GXV-Ada Initialization and Creation Subprograms

GXV-Ada initializes an instance for each window in the GIL file. The instance initialization function is named:

interface_window`_Objects_Initialize`

where *interface* is the name of the user interface, and *window* is the name of the window being initialized. For example, GXV-Ada initializes an instance for the window `financials` in the interface `report` using the subprogram `Report_Financials_Objects_Initialize`.

Following the instance initialization subprogram (or subprograms) are creation subprograms, one per interface element. Each element creation subprogram is named:

interface_window_element`_Create`

where:

- *interface* is the name of the user interface
- *window* is the name of the window in which the element is located (which is the window itself if the function creates a window)
- *element* is the name of the element

For example, a creation function creating a control area named `controls1` in a window named `win` in an interface named `intrfce` would be created by the function `Intrfce_Win_Controls1_Create`.

Both instance initialization functions and element creation functions are defined in the visible part of the `_ui.a` file to allow flexibility in their calling order.

3.2.4 *The* _stubs.a *File*

The _stubs.a file contains multiple compilation units:

- A declaration of the main program (parameterless function)
- A corresponding body of the main program
- A package declaration for main program execution and callback routines
- A corresponding package body for main program execution and callback routines.

The _stubs.a file contains code for each of the connections contained in the GIL file. Subprograms are generated for the ExecuteCode and CallFunction connections. Callbacks are automatically generated if an object does not already have a handler. Trace PUT_LINE statements are generated when the subprograms are called so you can receive confirmation of the call. GXV-Ada writes to STANDARD_ERROR.

The main program calls a corresponding routine in the generated package where all main program activities are handled. These activities include calls to the various XView and Devguide initialization subprograms. You can add to the main program or you can use parts of the main program to add to your own main program.

GXV-Ada does not overwrite an existing _stubs.a file. Instead, it generates a new _stubs.a file and saves the previous _stubs.a file as _stubs.a.BAK.

For example, suppose you have added application code into myApp_stubs.a. You then use Devguide to modify your interface; this generates a new myApp_stubs.a file, saving the original to myApp_stubs.a.BAK.

3.2.5 *The* _stubs.a.BAK *File*

Every time GXV-Ada generates a new _stubs.a file, it also produces a backup of the existing _stubs.a file. This backup file has the extension _stubs.a.BAK. When GXV-Ada is next run, it overwrites the backup file with a new one.

3.2.6 *The* `.info` *File*

When you add help text to elements in a user interface, GXV-Ada puts the help text in the `.info` file. Do not alter this `.info` file manually; any changes you make will be lost upon subsequent invocations of GXV-Ada. Use the help text editor in Devguide to change the help text.

3.2.7 *The* `Makefile` *File*

When GXV-Ada is run, it checks the current directory to see if a `Makefile` already exists. If a `Makefile` exists, then GXV-Ada does not generate a new `Makefile`. This feature protects a custom `Makefile`, ensuring that GXV-Ada does not write over or modify it. To add new libraries or flags to an existing `Makefile`, you must edit the `Makefile` directly.

After GXV-Ada generates its own `Makefile`, you can easily customize it to use your own custom code files as source files during compilation. The `Makefile` lists the source files in a section labeled `Parameters`, usually found near the top of the file. The parameter of interest is `SOURCES`. Add the file names of your custom files to the `SOURCES` parameter and `make` will compile them.

3.2.8 *Command-Line Options*

GXV-Ada provides several command-line options to enhance or customize code generation. These are described in Table 3-2.

Table 3-2 GXV-Ada Command-Line Options

Option	Description
`-s (-silent)`	Silent mode; no trace messages are generated..
`-v (-verbose)`	Verbose mode; a message is generated as each new user interface element is encountered.
`-n (-nomerge)`	Prohibits merging of existing and new `_stubs.a` files. This is the default behavior for this version.
`-p (-project)`	Generates code for a project. You must specify a project name.

Table 3-2 GXV-Ada Command-Line Options *(Continued)*

Option	Description
`-m (-main)`	Generates source for *projname*`.a` and *projname*`_b.a` files. Only works with the `-p` option present. Use the `-m` option if you have already run GXV-Ada on `.G` files comprising a project.
`-h (-helpfile)`	Generates only the help file text (the `.info` file).
`-? (-help)`	Prints out a help message explaining all GXV-Ada options.

You can use as many of these options as you want when you run GXV-Ada. To do so, type:

```
% gxv_ada options filename
```

3.3 Before Compiling

Before compiling, you must have the `ADAHOME` environment variable set to point to the SPARCompiler Ada release directory. “ADAHOME Environment Variable” on page 2-3 describes how to set this variable.

If you have custom Ada source code files, edit the `Makefile` to include these files in the `Parameters` section. Enter the names of all your Ada source code files in the `SOURCES` line.

3.4 Compiling and Linking

After your `Makefile` contains all of the associated source code files, you compile them by entering the command `make`. This program runs according to the contents of the `Makefile`. It:

- Checks to see if an Ada program library exists, and if not, creates one.
- Checks all files specified in the `SOURCES` parameter to see if any are out-of-date with respect to the last compile.
- Compiles and links all the out-of-date source code files.

You should run GXV-Ada to generate new files for changed GIL files before each compile. Any saved changes you make to an interface using Devguide will show up in the executable file after the compile. You need not alter GXV-Ada-generated files: `make` detects that files have changed, and GXV-Ada overwrites the files.

The compiled code is placed in the file named in the PROGRAM parameter of the Makefile. To run it, type this file name in the Shell.

3.5 Generating Ada Source Code: An Example

Below are the steps required to generate Ada source code and run a demonstration produced by Devguide and GXV-Ada.

1. **Log on to a workstation and run OpenWindows.**
   Make sure that the environment variables $ADAHOME, $OPENWINHOME, $GUIDEHOME, and $LD_LIBRARY_PATH are properly set.

2. **Create an interface in Devguide.**
   Use Devguide to define an interface. In the rest of this section, an existing GIL file is used.

   The example file is:
   $ADAHOME/examples/gxv_ada_examples/button2/button2.G

   If you choose to create and save your own interface, do so at this time using Devguide. Otherwise, copy the button2.G GIL file to a new working directory.

3. **Invoke GXV-Ada on the GIL file to generate the code.**
   Type:

   ```
   % gxv_ada button2
   ```

   GXV-Ada then generates the interface code and the Makefile.

4. **Examine the generated Makefile and interface code.**
   The Makefile generated by GXV-Ada should be displayed as:

```
# This file was generated by ‘gxv_ada’ from ‘button2.G’.

# Parameters.

PROGRAM = button2

# Derived parameters.

SOURCES = $(PROGRAM)_ui.a $(PROGRAM)_stubs.a $(PROGRAM)_ui_b.a
```

(Continued)

```
# Compiler flags.

ADAMAKEFLAGS = -v
ADALDFLAGS = -o $(PROGRAM)

# Standard targets.

all: $(PROGRAM)
sources: $(SOURCES)
adamakelib = ada.lib

$(PROGRAM)_ui.a: $(adamakelib)
$(PROGRAM)_stubs.a: $(PROGRAM)_ui.a
$(PROGRAM)_ui_b.a: $(PROGRAM)_stubs.a

$(adamakelib):
     a.mklib . $(ADAHOME)/xview
     a.path -i $(ADAHOME)/libguidexv_iface

adamake: $(adamakelib) $(SOURCES)
     a.make $(ADAMAKEFLAGS) -f $(SOURCES)

$(PROGRAM): adamake
     a.ld $(ADALDFLAGS) $(PROGRAM)

adaclean: ada.lib
     a.cleanlib
     $(RM) .options

adarmlib: ada.lib
     a.rmlib
     $(RM) .options

clean:
     $(RM) .options core *.delta *.BAK $(PROGRAM)_ui.a
       $(PROGRAM)_ui_b.a $(PROGRAM).info
     a.rmlib -f

veryclean:
     $(RM) .options Makefile core $(PROGRAM)*.BAK
       $(PROGRAM)*.delta $(PROGRAM)*.tmp $(PROGRAM) $(SOURCES)
       $(PROGRAM).info
     a.rmlib -f
```

The `button2_ui.a` file for GXV-Ada should be displayed as:.

```
--
-- button2_ui.a - User interface object and function declarations.
-- This file was generated by ‘gxv_ada’ from ‘button2.G’.
-- DO NOT EDIT BY HAND.
--

with Xvi_Attr; use Xvi_Attr;
with Xvi_Base; use Xvi_Base;
package Button2_Ui is
     Instance : Attr_Attribute;

     type button2_window1_Objects is
       record
         Window1 : Xv_Opaque;
         Controls1 : Xv_Opaque;
         Fto : Xv_Opaque;
       end record;

     type button2_window1_Objects_Ptr is access
       button2_window1_Objects;

     function button2_window1_Objects_Initialize (
       Ip    : in button2_window1_Objects_Ptr;
       Owner : in Xv_Opaque)
       return button2_window1_Objects_Ptr;

     function button2_window1_Window1_Create (
       Ip    : in button2_window1_Objects_Ptr;
       Owner : in Xv_Opaque) return Xv_Opaque;

     function button2_window1_Controls1_Create (
       Ip    : in button2_window1_Objects_Ptr;
       Owner : in Xv_Opaque) return Xv_Opaque;

     function button2_window1_Fto_Create (
       Ip    : in button2_window1_Objects_Ptr;
       Owner : in Xv_Opaque) return Xv_Opaque;
end Button2_Ui;
```

The `button2_ui_b.a` and `button2_stubs.a` files are more complex and are not duplicated here, though they follow the same style as the previous source code example. An abbreviated GXV-Ada `button2_stubs.a` listing is provided in step 7.

5. **Type `make` to build a test copy of the interface.**
   The `make` command builds an executable named `button2`. Be sure to set the environment variables `$ADAHOME` and `$LD_LIBRARY_PATH` properly.

6. **Type `button2` to run the executable.**
   Each time you press the button displayed in the base window, the application prints the connection name to the console.

   To quit the application, choose Quit from the base window menu.

7. **Modify the `button2_stubs.a` file.**
   The file originally generated by GXV-Ada contains a simple `Print_Msg` procedure that is invoked whenever the button is pressed. Modify the package body `Button2_Callbacks` to include a global integer counter whose value is displayed on the console each time the `Print_Msg` procedure is invoked.

Your solution for GXV-Ada should resemble the following:

```
--
-- button2_stubs.a - Notify and event callback function stubs.
-- This file was generated by ‘gxv_ada’ from ‘button2.G’.
-- DO NOT EDIT BY HAND.
--

with Gdd; use Gdd;
...
with Button2_Ui; use Button2_Ui;
package Button2_Callbacks is
  function Button2_Main return INTEGER;
  procedure Print_Msg (
    Item  : in Xvi_Panel.Panel_Item;
    Event : in Event_Ptr);
end Button2_Callbacks;

function Button2 return INTEGER;  -- The main program.
...
with UNCHECKED_CONVERSION;
package body Button2_Callbacks is
  --
  -- Global object definitions.
  --

  Button2_window1 : button2_window1_Objects_Ptr;
  GLOBAL_COUNT : NATURAL := 0;  -- NEWLY ADDED

  ...
  --
  -- Notify callback function for ‘fto’.
  --

  procedure Print_Msg (
    Item  : in Xvi_Panel.Panel_Item;
    Event : in Event_Ptr) is

    function To_button2_window1_Objects_Ptr is new
      UNCHECKED_CONVERSION (
      Caddr_T, button2_window1_Objects_Ptr);
```

(Continued)

```
    Ip : button2_window1_Objects_Ptr:=
      To_button2_window1_Objects_Ptr (
        Xv_Get (Xvi_Panel.Panel (Item),
            (Xv_Key_Data, INTEGER (Instance), Xv_Opaque_Null)));

  begin
    GLOBAL_COUNT := GLOBAL_COUNT + 1;
    PUT_LINE (STANDARD_ERROR,
      "button2: Print_Msg button pressed" &
      INTEGER'IMAGE (GLOBAL_COUNT) & " times.");

    -- gxv_start_connections DO NOT EDIT THIS SECTION

    -- gxv_end_connections

  end Print_Msg;
end Button2_Callbacks;
```

The variable *GLOBAL_COUNT* was added near the top of the package body, incremented in `Print_Msg`, and used in the output statement of the `Print_Msg` procedure.

8. **Type** `make` **to build the complete application.**
   SPARCompiler Ada recompiles the necessary interface code and rebuilds an executable called `button2`.

9. **Rerun** `button2` **and observe the new output in the console window at every button press.**
   The output should now indicate the number of times the button has been pressed since the application was invoked.

   To quit the application, choose Quit from the base window menu.

3.6 Directories Included with GXV-Ada

The following directories are included with GXV-Ada:

`$ADAHOME/libguidexv_iface`
: This directory contains Ada-compiled and C- compiled code that is required by applications generated by GXV-Ada. The directory contains a variety of capabilities used to bridge Ada code with the underlying C-based OpenWindows environment.

`$ADAHOME/examples/gxv_ada_examples`
: This directory contains a wide variety of uncompiled Devguide GIL files and supporting text and icon files. You can run these files through GXV-Ada to observe various Devguide features.

Ada-Specific Information 4

4.1 Comparing GXV-Ada with GXV

This chapter describes the main differences between GXV-Ada and GXV. Because GXV-Ada is derived from GXV, which is XView-based, the differences are few. This chapter also discusses how GXV-Ada functions with some of the major Ada routines.

4.1.1 Ada XView Bindings

GXV-Ada targets its interface source code generation to the SPARCompiler Ada XView binding located in the directory `$ADAHOME/xview`. GXV, on the other hand, targets its code generation directly to the C definition of the XView toolkit. GXV-Ada is one level removed, generating code to the Ada XView. Its bindings in turn communicate with the C-level XView routines.

The Ada XView binding is not a simple transliteration of C to Ada. Because of the complexities introduced in the Ada XView bindings, some GXV-Ada-generated code differs greatly from GXV-generated code. The functionality of the underlying user interface application remains the same, however.

4.1.2 Callbacks

GXV uses the name of the C function as the callback parameter. GXV-Ada must use the `ADDRESS` attribute notation prefixed by the callback subprogram name to satisfy Ada compilation rules. The end result is the same for both GXV and GXV-Ada, that is, callback routines are invoked in a similar fashion.

The types of the parameters and result of a GXV-Ada callback subprogram match closely those of the GXV callbacks. Usually, full-word address values or scalars are passed to GXV-Ada callbacks and type checking is avoided, as it would be in C.

4.1.3 The `Makefile` File

Because of the different compilation and linking requirements of a C versus an Ada program, GXV-Ada generates its own Ada-specific `Makefile`. Chapter 3, “Generated Source Code,” includes an example.

The GXV-Ada `Makefile` accounts for formulation of an Ada program library and relies on the SPARCompiler Ada `a.make` program to properly compile the interface code. Ada dependencies among the interface code files are accounted for so that recompilation by means of executing `make` can be achieved.

4.1.4 The Main Program

C and Ada have different requirements for the name and format of a main program. GXV-Ada creates a main program that is a parameterless function, following the UNIX model of programs that return status codes. However, unlike C, GXV-Ada names its main programs the same as the prefix of the GIL file. Doing so allows for multiple compilations of GXV-Ada-generated programs in a single Ada program library.

The `_stubs.a` file contains the main program. The main program is compiled with a separate declaration and body to allow for better recompilation behavior if you modify the `_stubs.a` file.

The main program invokes a routine of the same name in the callbacks package, also housed in the `_stubs.a` file. This level of indirection allows GXV-Ada to be more compliant with the manner in which GXV generates its code in the `_stubs.a` file.

4.1.5 Unsupported GXV Features

This release of GXV-Ada does not support all capabilities of its fellow Devguide program, GXV. This section identifies those GXV features which GXV-Ada does not support.

- File merging in GXV-Ada is supported, but follows a different paradigm than its GXV counterpart due to differences between the generated `stubs` files for each
- No internationalization support.

4.1.6 Special GVX-Ada Features

GXV-Ada supports a few features that GXV does not support. These features are detailed in this section.

- The `Makefile` supports `make veryclean`, which performs a more careful and thorough removal of GXV-Ada-generated files than does the `make clean` capability of either GXV-Ada or GXV
- The `Makefile` supports `make adaclean`, which removes all Ada compilations from the Ada program library
- The `Makefile` supports `make adarmlib`, which removes the Ada program library and the `.options` file, if one exists
- GXV-Ada names its main program the same as the prefix of the GIL file instead of `main`, as GXV is required to do.

4.2 Compiling and Linking

Ada is more rigorous in its compilation requirements than is C. This necessarily requires GXV-Ada to provide more compilation-time mechanisms than the Devguide C code generators.

A GXV-Ada application `Makefile` includes capabilities for establishing an Ada program library, inclusion of required search paths in the library, and proper compilation and linking order. GXV-Ada uses the library manipulation tools of SPARCompiler Ada to accomplish these tasks, as follows:

- `a.mklib` establishes an Ada program library

- `a.path` inserts appropriate Ada program libraries onto the current search path
- `a.make` compiles the interface code
- `a.ld` links an interface executable file

See the SPARCworks Ada compiler documentation for details.

4.3 *Editing the* `_stubs.a` *File*

The `_stubs.a` file contains four program units:

- Package declaration for the callbacks and main program operation code
- Declaration of the Ada main program
- Body of the Ada main program
- Package body of the callbacks and main program operation code

Most changes to the `_stubs.a` file are focused on the package body, where callback stubs are generated. Do not change the main program, but only make additions to the package declaration; for example, you can safely add a new callback subprogram.

4.4 *Recompiling*

Ada recompilations are automatically factored into the construction of the `Makefile`. In most cases, you need only execute `make` to cause a recompile and relink the user interface source code. In cases where you have incorporated new files into the interface code or have created additional dependencies to outside Ada program libraries, you must change the `Makefile`.

You can account for new file names in the `Makefile` simply by editing the definition of the `SOURCES` parameter. Do the same for new dependencies to Ada program libraries by changing the options on the `ADAMAKEFLAGS` parameter. Including the `-All` option for `ADAMAKEFLAGS` can effectively cause recompilation of outside Ada program libraries when warranted.

4.5 Debugging

Debugging a GXV-Ada application is the same as debugging any other Ada program. The SPARCompiler Ada `a.db` utility performs exactly as it does for any other Ada program.

Index

F

G

H

I

L

M

N