Introduction 1

This chapter provides an overview of Guide to the OPEN LOOK Intrinsics Toolkit (Golit).

Devguide and Golit

Devguide is a development tool that enables you to create and test user interfaces without writing any code.

Devguide produces GIL (Guide Interface Language) files that you can convert into interface toolkit code with a code generator.

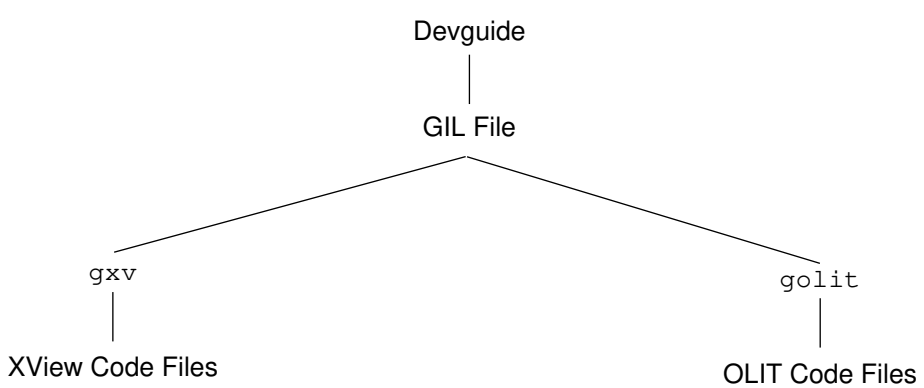

Figure 1-1 Devguide Tools and Files

There are two code generators described in the current manual set (see Figure 1-1). This manual describes the Golit code generator for the OLIT toolkit.

Names and Terminology

The word *Guide* in the full name, *OpenWindows Developer's Guide*, is an acronym for Graphical User Interface Design Editor. The abbreviated name, *Devguide*, refers to the graphical interface tool you use to develop the user interface.

The term *Golit* refers to the code generator that takes the GIL file produced by Devguide and produces the executable OLIT code for your user interface. This code is referred to as the *UI* (user interface) code, which contrasts with the *application code*.

Installing Devguide and Golit

To install Devguide and the code generators, follow the instructions in the *Software Developer Kit Installation Guide*.

Before you use Devguide and Golit, make sure the environment variables `GUIDEHOME` and `OPENWINHOME` point to Devguide's and OpenWindows' home directories respectively. These environment variables must be set correctly for Devguide, Golit, and the generated Makefile to work.

How you interact with Devguide and Golit

The basic interaction with Devguide is very simple. You use Devguide to develop your user interface with the OPEN LOOK look and feel by dragging and positioning glyphs on interface windows. Devguide saves a description of the interface in a GIL file. This process is explained in detail in the *OpenWindows Developer's Guide: User's Guide.*

Once you have built a working prototype of your user interface and generated the GIL file describing your prototype, you run the Golit program on the GIL file, shown below as `<fn>.G`. The `.G` suffix is the standard suffix for GIL files. Golit produces the three files shown in Figure 1-2.

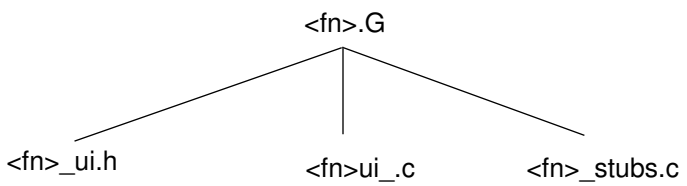

Figure 1-2 GolitGenerated Source Files

Golit also automatically generates a program makefile.

Devguide and Golit allow you to organize several GIL files into projects. Devguide saves projects in files that have a `.P` extension. When you run Golit on a project file, it generates the files shown in Figure 1-2 for each `.G` file in the project. It also generates `<projectname>.c` and `<projectname>.h` files.

How Golit and libgolit Work

Golit-generated code uses the OLIT toolkit and a runtime library (called libgolit.a) to create user interfaces. Golit-generated code is different from OLIT user interface code that you might write from scratch. It does not directly call widget creation routines. Instead, it initializes data structures to describe the user interface and form a *widget tree*. It then calls libgolit routines to traverse this tree and perform the actual widget creation.

Golit's design isolates code that describes interface objects from code that instantiates and manipulates the objects. Golit generates separate files for each type of code. This makes it easy to modify an application. It's easy to change the application code, since it's all in one place; you don't have to wade through detailed UI code to find it. It's also easy to change an interface and regenerate UI code, since you don't have to worry about overwriting application code. Golit only overwrites the files that contain the interface description.

Golits default widget creation does not allow you to change widget resources when a widget is created. However, Golit provides a way around this. You can use a special type of callback, called a *Create Proc*, to intercept the default widget creation and provide your own widget creation code. The Create Proc is invoked whenever the widget is about to be created. It can change resources at creation time as needed.

1

The libgolit library provides utility functions that enable you to selectively instantiate the widget tree. You can instantiate an individual widget or entire portions of that widget tree once, or many times. This allows you to reuse parts of the interface description that you created in Devguide, making your code more compact and efficient.