Getting Started with GMF 2

This chapter describes how to generate simple user interfaces and integrate them with your applications. This chapter is intended to be a "quick start guide" that provides you with the minimal instructions necessary to start using `gmf`. The topics discussed here are covered in greater detail in Chapter 3, "GMF Functionality in Detail."

Note – Although this chapter discusses the creation of a Devguide user interface from scratch, remember that Devguide produces Open Look output and it is unlikely that you will want to use Devguide to create a user interface if the final application will be a Motif application. If, despite this warning, you decide to create a new Devguide interface for a Motif application, read the *OpenWindows Developer's Guide: User's Guide*, setting the Toolkit to OLIT if you plan on doing connections (note, however, that some connections may not work for a Motif application).

If you already have a Devguide-created GIL file, you can use `gmf` to convert the GIL file to Motif C code.

2

Summary of How to Create an Application with GMF

To create an application with Devguide and `gmf`, follow these steps:

1. **Generate Devguide UI code by running `gmf` on a GIL or project file.**
   `gmf` generates three C files and a makefile for each GIL file. The C files contain the interface source code. The makefile contains the Make utility commands to build the interface. If you run `gmf` on a project file, it generates two additional C files and a supplement to the makefile.

2. **Use `make` to build a test copy of the interface.**
   `gmf` generates all the code necessary to create an executable, so you can compile the code by itself immediately after you generate it. It is recommended that you test the interface before you add your application code.

3. **Insert your application code in the callback function templates generated by `gmf` in the `_stubs.c` file.**
   Your application code can change the resources of widgets in the interface or it can instantiate additional widgets.

4. **Run `make` again to build the application.**
   The Make utility compiles the interface and your application code into a complete executable.

Generating User Interface Code from GIL Files

To generate code from a GIL file, run `gmf` as follows: :

```
% gmf <GIL_filename>
```

If you are generating code for a project, run `gmf` on the project file by entering the following command:

```
% gmf -p <project_filename>
```

It is not necessary to type the `.G` or the `.P` filename extension after the GIL or project file filenames, since `gmf` adds the appropriate extension automatically.

GMF-Generated Files

gmf-generated files contain all the source code necessary to create an executable that uses the Motif toolkit and the libgmf runtime library.

When you run gmf on a single GIL file, it generates the following files in the current working directory:

- <GIL_filename>_stubs.c - skeleton main program for the executable. It includes the callback templates that you insert your application code into.
- <GIL_filename>_ui.c - code that defines the user interface objects. Unless you are writing advanced applications, you don't need to look at this file.
- <GIL_filename>_ui.h - header file that declares the user interface objects, external callbacks, and external creation procedures. Normally, you won't need to look at this file.
- Makefile - a template makefile to build the executable. Makefile contains make utility commands to compile your interface and any application code you include in the _stubs.c file
- <GIL_filename>.resource - resource file created by -r (all resources) or -i (internationalization) flags.

The filenames of these files are based on the name of the GIL file. For example, if you use gmf to generate source code for a GIL file named display.G, gmf creates the files, display_ui.c, display_ui.h, and display_stubs.c.

When you run gmf on a project file, it generates the files listed above (except for the makefile) for each GIL file in the project. It also generates the following files:

- <project_name>.c - includes the main program which is left out of the individual _stubs.c files. It also includes templates for callbacks shared by the GIL files in the project.
- <project_filename>.h - external declarations for callbacks shared by the GIL files.
- <project_filename>.make - addition to the makefile that is automatically included.

The filenames of these files are based on the name of the project file. For example, if you use `gmf` to generate source code for a project file named `myproject.P`, `gmf` creates the files `myproject.c`, `myproject.h`, and `myproject.make`.

Compiling and Testing Interface Code

To compile the interface code generated by `gmf`, type the following in the appropriate directory after you have generated the code:

```
% make
```

The `make` utility compiles and links the `gmf`-generated code. The resulting executable has the same name as the GIL file, without the `.G` filename extension. If you have generated code for a project, it will have the same name as the project file without the `.P` extension.

Note – Devguide provides a *test mode* that allows you to test an interface while you are designing it. However, some connections that specify events or actions specific to `gmf` do not work in the Devguide test mode. Also, since Devguide displays an OPEN LOOK look and feel, the interface will look different in the test mode than it will in the compiled application. Always test your interface by compiling the generated interface code once and running it before you add your application code.

The executable you create when you compile the generated code provides a complete working model of the user interface. When you perform an action in the interface (for example, a button press) the program prints the name of any connections associated with that action to the console.

Integrating Application Code with GMF Interface Code

To add your application code to the `gmf` interface code, modify the callback templates in the `_stubs.c` file. If you call functions that you keep in separate files, you also need to modify the `Makefile` to compile these files. Use XtVaSetValues or XtVaGetValues to set or get resources, as you normally would.

Caution – Do not alter the `_ui.c` or `_ui.h` files. If you add any code to these files, you will lose it the next time you run `gmf`.

Regenerating Code for a Modified Interface

As noted above, it is unlikely that you will want to modify your Devguide interface after running `gmf`; if you do, however, `gmf` will do the following each time it is run:

- Overwrite the existing `_ui.c` and `_ui.h` files.
- Back up the current `_stubs.c file` to `_stubs.c.BAK`.
- Generate a new `_stubs.c` file that contains templates for all the connections specified in the GIL file.
- Merge any code that you inserted in the original `_stubs.c` into the new `_stubs.c`.
- Create a `_stubs.c.delta` file that tells you how it has changed `_stubs.c`. This file lists the added text and the affected line numbers.

If you are regenerating code for a *project*, `gmf` will perform the steps above for each of the GIL files in the project. It will also do the following:

- Back up the `<projectname>.c` and `<project_name>.h` files to `.BAK` files.
- Generate a new `<projectname>.c` file and merges it with the old one, listing the changes in a `.delta` file.
- Overwrite the original `<project_name>.h` file.

2