This chapter contains material on topics of interest to advanced Igor users.

Procedure Modules

Igor supports grouping procedure files into modules to prevent name conflicts and to isolate procedure packages from other packages. This feature is for intermediate to advance Igor programmers.

There are three types of modules:

- ProcGlobal
- Regular Modules
- Independent Modules

You set a procedure file's module using the **ModuleName** or **IndependentModule** pragmas. A procedure file that does not contain one of these pragmas is in the ProcGlobal module by default.

Igor compiles independent modules separately from other procedures. This allows independent modules to continue to work even if there are errors in other procedure files. Independent module programming is demanding and is intended for use by advance Igor programmers. See **Independent Modules** on page IV-238 for details.

Regular modules provide protection against name conflicts and are easier to work with than independent modules. Regular modules are intended for use by intermediate to advanced Igor programmers who are creating procedure packages. See **Regular Modules** for details.

Regular Modules

Regular modules provide a way to avoid name conflicts between procedure files. Regular modules are distinct from "independent modules" which are discussed in the next section.

Igor's module concept provides a way to group related procedure files and to prevent name conflicts between procedure packages. You specify that a procedure file is part of a regular module using the #pragma ModuleName statement. The statement is allowed in any procedure file but not in the built-in main procedure window.

A procedure file that does not contain a #pragma ModuleName statement (or a #pragma IndependentModule statement) defines public procedure names in the default ProcGlobal module. All functions in the procedure file can be called from any other procedure file in ProcGlobal or in any regular module by simply specifying the name without any name qualifications (explained below).

When you execute a function from the command line or use the **Execute** operation, you are operating in the ProcGlobal context.

Functions are public by default and private if declared using the static keyword. For example

```
// In a procedure file with no #pragma ModuleName or #pragma IndependentModule

static Function Test()           // "static" means private to procedure file
   Print "Test in ProcGlobal"
End

Function TestInProcGlobal()      // Public
   Print "TestInProcGlobal in ProcGlobal"
End
```

Because it is declared static, the Test function is private to its procedure file. Each procedure file can have its own static Test function without causing a name conflict.