Programming Overview

Overview

You can perform powerful data manipulation and analysis interactively using Igor's dialogs and the command line. However, if you want to automate common tasks or create custom data analysis features, then you need to use procedures.

You can write procedures yourself, use procedures supplied by WaveMetrics, or find someone else who has written procedures you can use. Even if you don't write procedures from scratch, it is useful to know enough about Igor programming to be able to understand code written by others.

Programming in Igor entails creating procedures by entering text in a procedure window. After entering a procedure, you can execute it via the command line, by choosing an item from a menu, or using a button in a control panel.

The bulk of the text in a procedure window falls into one of the following categories:

- Pragmas, which send instructions from the programmer to the Igor compiler
- Include statements, which open other procedure files
- Constants, which define symbols used in functions
- Structure definitions, which can be used in functions
- Proc Pictures, which define images used in control panels, graphs, and layouts
- Menu definitions, which add menu items or entire menus to Igor
- Functions — compiled code which is used for nearly all Igor programming
- Macros — interpreted code which, for the most part, is obsolete

Functions are written in Igor's programming language. Like conventional procedural languages such as C or Pascal, Igor's language includes:

- Data storage elements (variables, strings, waves)
- Assignment statements
- Flow control (conditionals and loops)
- Calls to built-in and external operations and functions
- Ability to define and call subroutines

Igor programming is easier than conventional programming because it is much more interactive — you can write a routine and test it right away. It is designed for interactive use within Igor rather than for creating stand-alone programs.

Organizing Procedures

Procedures can be stored in the built-in Procedure window or in separate auxiliary procedure files. Chapter III-13, **Procedure Windows**, explains how to edit the Procedure window and how to create auxiliary procedure files.

At first you will find it convenient to do all of your Igor programming in the built-in Procedure window. In the long run, however, it will be useful to organize your procedures into categories so that you can easily find and access general-purpose procedures and keep them separate from special-case procedures.

This table shows how we categorize procedures and how we store and access the different categories.

Category	What	Where	How
Experiment Procedures	These are specific to a single Igor experiment. They include procedures you write as well as window recreation macros created automatically when you close a graph, table, layout, control panel, or Gizmo plot.	Usually experiment procedures are stored in the built-in Procedure window. You can optionally create additional procedure windows in a particular experiment but this is usually not needed.	You create an experiment procedure by typing in the built-in Procedure window.
Utility Procedures	These are general-purpose and potentially useful for any Igor experiment. WaveMetrics supplies utility procedures in the WaveMetrics Procedures folder. You can also write your own procedures or get them from colleagues.	WaveMetrics-supplied utility procedure files are stored in the WaveMetrics Procedures folder. Utility procedure files that you or other Igor users create should be stored in your own folder, in the Igor Pro User Files folder (see **Igor Pro User Files** on page II-31 for details) or at another location of your choosing. Place an alias or shortcut for your folder in "Igor Pro User Files/User Procedures".	Use an include statement to use a WaveMetrics or user utility procedure file. Include statements are described in **The Include Statement** on page IV-155.
Global Procedures	These are procedures that you want to be available from all experiments.	Store your global procedure files in "Igor Pro User Files/Igor Procedures" (see **Igor Pro User Files** on page II-31 for details). You can also store them in another folder of your choice and place an alias or shortcut for your folder in "Igor Pro User Files/Igor Procedures".	Igor automatically opens any procedure file in "Igor Pro 7 Folder/Igor Procedures" and "Igor Pro User Files/Igor Procedures" and subfolders or referenced by an alias or shortcut in those folders, and leaves it open in all experiments.

Following this scheme, you will know where to put procedure files that you get from colleagues and where to look for them when you need them.

Utility and global procedures should be general-purpose so that they can be used from any experiment. Thus, they should not rely on specific waves, global variables, global strings, specific windows or any other objects specific to a particular experiment. See **Writing General-Purpose Procedures** on page IV-156 for further guidelines.

After they are debugged and thoroughly tested, you may want to share your procedures with other Igor users via **IgorExchange**.

WaveMetrics Procedure Files

WaveMetrics has created a large number of utility procedure files that you can use as building blocks. These files are stored in the WaveMetrics Procedures folder. They are described in the WM Procedures Index help file, which you can access through the Help→Help Windows menu.

You access WaveMetrics procedure files using include statements. Include statements are explained under **The Include Statement** on page IV-155.

Using the Igor Help Browser, you can search the WaveMetrics Procedures folder to find examples of particular programming techniques.

Macros and Functions

There are two kinds of Igor procedures: **macros** and **functions**. They use similar syntax. The main difference between them is that Igor compiles user functions but interprets macros.

Because functions are compiled, they are dramatically faster than macros. Compilation also allows Igor to detect errors in functions when you write the function, whereas errors in macros are detected only when they are executed.

Functions provide many programming features that are not available in macros.

Macros are a legacy of Igor's early days. With rare exceptions, **all new programming should use functions**, not macros. To simplify the presentation of Igor programming, most discussion of macros is segregated into Chapter IV-4, **Macros**.

Scanning and Compiling Procedures

When you modify text in a procedure window, Igor must process it before you can execute any procedures. There are two parts to the processing: scanning and function compilation. In the scanning step, Igor finds out what procedures exist in the window. In the compilation step, Igor's function compiler converts the function text into low-level instructions for later execution.

For the sake of brevity, we use the term "compile" to mean "scan and compile" except when we are specifically pointing out the distinction between these two steps.

You can *explicitly* compile the procedures using the Compile button in the Procedure window or the Compile item in the Macros menu.

By default, Igor *automatically* compiles the procedure text at appropriate times. For example, if you type in the Procedure window and then hide it by clicking in the close button, Igor will automatically compile.

If you have many procedures that take long to compile, you may want to turn auto-compiling off using the Macros menu.

When Auto-compile is deselected, Igor compiles only when you explicitly request it. Igor will still scan the procedures when it needs to know what macros and functions exist.

Indentation Conventions

We use indentation to indicate the structure of a procedure.

```
                        Function Example()
                            <Input parameter declarations>

The body of the function
is indented by one tab.     <Local variable declarations>

                            if (condition)
                                <true part>          Indentation clearly shows what is
                            else                      executed if the condition is true and
                                <false part>          what is executed if it is false.
                            endif

                            do
                                <loop body>          The body of the loop is
                            while (condition)        indented by one tab.
                        End
```

The structural keywords, shown in bold here, control the flow of the procedure. The purpose of the indentation is to make the structure of the procedure apparent by showing which lines are within which structural keywords. Matching keywords are at the same level of indentation and all lines within those keywords are indented by one tab.

The Edit menu contains aids for maintaining or adjusting indentation. You can select multiple lines and choose Indent Left or Indent Right. You can have Igor automatically adjust the indentation of a procedure by selecting the whole procedure or a subset and then choosing Adjust Indentation.

Igor does not require that you use indentation but we recommend it for readability.

What's Next

The next chapter covers the core of Igor programming — writing user-defined functions.

Chapter IV-4, **Macros**, explains macros. Because new programming does not use macros, that chapter is mostly of use for understanding old Igor code.

Chapter IV-5, **User-Defined Menus**, explains user-defined menus. It explains how you can add menu items to existing Igor menus and create entire new menus of your own.

Chapter IV-6, **Interacting with the User**, explains other methods of interacting with the user, including the use of dialogs, control panels, and cursors.

Chapter IV-7, **Programming Techniques**, covers an assortment of programming topics. An especially important one is the use of the include statement, which you use to build procedures on top of existing procedures.

Chapter IV-8, **Debugging**, covers debugging using Igor's symbolic debugger.

Chapter IV-9, **Dependencies**, covers dependencies — a way to tie a variable or wave to a formula.

Chapter IV-10, **Advanced Topics**, covers advanced topics, such as communicating with other programs, doing FTP transfers, doing data acquisition, and creating a background task.