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.

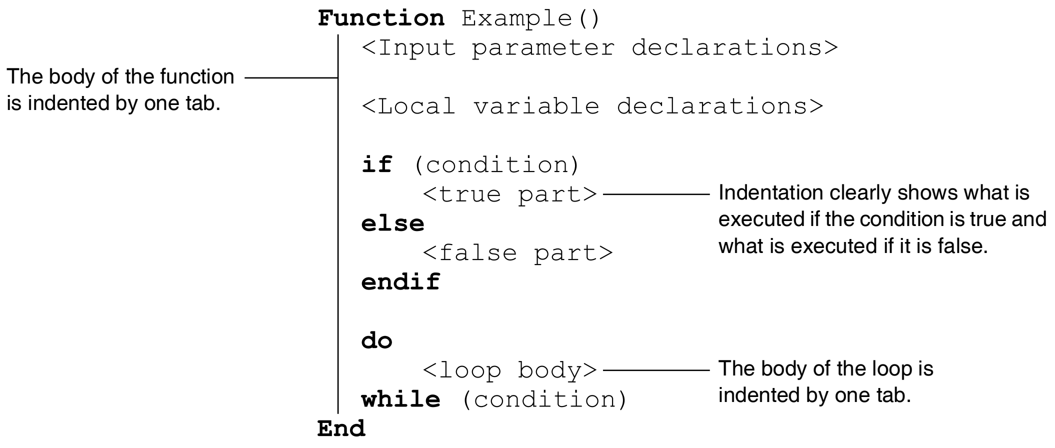
By default, Igor *automatically* compiles the procedure text at appropriate times. For example, if you type in the Procedure window and then hide it by clicking in the close button, Igor will automatically compile.

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>

	<Local variable declarations>

	if (condition)
		<true part>
	else
		<false part>
	endif

	do
		<loop body>
	while (condition)
End
```

The body of the function is indented by one tab.
Indentation clearly shows what is executed if the condition is true and what is executed if it is false.
The body of the loop is indented by one tab.

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 struc-