The TestInProcGlobal function is public so there can be only one public function with this name in Proc-Global.

In this example the static Test function is accessible only from the procedure file in which it is defined.

Sometimes you have a need to avoid name conflicts but still want to be able to call functions from other procedure files, from control action procedures or from the command line. This is where a regular module is useful.

You specify that a procedure file is in a regular module using the ModuleName pragma so that even static functions can be called from other procedure files. For example:

```
#pragma ModuleName = ModuleA      // The following procedures are in ModuleA

static Function Test()            // Semi-private
   Print "Test in ModuleA"
End

Function TestModuleA()            // Public
   Print "Test in ModuleA"
End
```

Because it is declared static, this Test function does not conflict with Test functions in other procedure files. But because it is in a regular module (ModuleA), it can be called from other procedure files using a qualified name:

```
ModuleA#Test()                    // Call Test in ModuleA
```

This qualified name syntax overrides the static nature of Test and tells Igor that you want to execute the Test function defined in ModuleA. The only way to access a static function from another procedure file is to put it in a regular module and use a qualified name.

If you are writing a non-trivial set of procedures, it is a good idea to use a module and to declare your functions static, especially if other people will be using your code. This prevents name conflicts with other procedures that you or other programmers write.

Make sure to choose a distinctive module name. Module names must be unique.

Regular Modules in Action Procedures and Hook Functions

Control action procedures and hook functions are called by Igor at certain times. They are executed in the ProcGlobal context, as if they were called from the command line. This means that you must use a qualified name to call a static function as an action procedure or a hook function. For example:

```
#pragma ModuleName = RegularModuleA

static Function ButtonProc(ba) : ButtonControl
   STRUCT WMButtonAction &ba

   switch (ba.eventCode)
      case 2: // mouse up
         Print "Running RegularModuleA#ButtonProc"
         break
   endswitch

   return 0
End

static Function CreatePanel()
   NewPanel /W=(375,148,677,228)
   Button button0,pos={106,23},size={98,20},title="Click Me"
```