A procedure file that is designed to be #included should ideally work inside or outside of an independent module. Read the sections on independent modules below to learn what the issues are.

When developing an independent module, you will usually want to execute:

```
SetIgorOption IndependentModuleDev=1
```

Independent Modules and #include

If you #include a procedure file from an independent module, Igor copies the #included file into memory and makes it part of the independent module by inserting a #pragma IndependentModule statement at the start of the copy. If the same file is included several times, there will be several copies, each with a different independent module name.

Warning: Do not edit procedure windows created by #including into an independent module because they are temporary and your changes will not be saved. You would not want to save them anyway because Igor has modified them.

Warning: Do not #include files that already contain a #pragma IndependentModule statement unless the independent module name is the same.

Limitations of Independent Modules

Independent modules are not for every-day programming and are more difficult to create than normal modules because of the following limitations:

1. Macros and Procs are not supported.
2. Button and control dialogs do not list functions in an independent module.
3. Functions in an independent module can not call functions in other modules except through the Execute operation.

Independent Modules in Action Procedures and Hook Functions

Normally you must use a qualified name to invoke a function defined in an independent module from the ProcGlobal context. Control action procedures and hook functions execute in the ProcGlobal context. But, as a convenience and to make #include files more useful, Igor eliminates this requirement when you create controls and specify hook functions from a user-defined function in an independent module.

When you execute an operation that creates a control or specifies a hook function while running in an independent module, Igor examines the specified control action function name or hook function name. If the named function is defined in the same independent module, Igor automatically inserts the independent module name. This means you can write something like:

```
#pragma IndependentModule = IndependentModuleA
Function SetProcAndHook()
   Button b0, proc=ButtonProc
   SetWindow hook(Hook1)=HookFunc
End
```

You don't have to write:

```
#pragma IndependentModule = IndependentModuleA
Function SetProcAndHook()
   Button b0, proc=IndependentModuleA#ButtonProc
   SetWindow hook(Hook1)=IndependentModuleA#HookFunc
End
```

Such independent module name insertion is only done when an operation is called from a function defined in an independent module. It is not done if the operation is executed from the command line or via **Execute**.

The control action function or hook function must be public (non-static) except as described under **Regular Modules Within Independent Modules** on page IV-242.