This nesting of modules is useful to prevent name conflicts in a large independent module project comprising multiple procedure files. Otherwise it is not necessary.

Because all procedure files in a given independent module are compiled separately from all other files, function names never conflict with those outside the group and there is little or no need to use the static designation on functions in an independent module. However, if need be, you can call static functions in a regular module inside an independent module from outside the independent module using a triple-qualified name:

```
IndependentModuleName#RegularModuleName#FunctionName()
```

## Calling Routines From Other Modules

Code in an independent module can not directly call routines in other modules and usually should not need to. If you must call a routine from another module, you can do it using the **Execute** operation. You must use a qualified name. For example:

```
Execute "ProcGlobal#foo()"
```

To call a function in a regular module, you must prepend ProcGlobal and the regular module name to the function name:

```
Execute "ProcGlobal#MyRegularModule#foo()"
```

Calling a nonstatic function in a different independent modules requires prepending just the other independent module name:

```
Execute "OtherIndependentModule#bar()"
```

Calling static functions in other independent modules requires prepending the independent module name and a regular module name:

```
Execute "OtherIndependentModule#RegularModuleName#staticbar()"
```

## Using Execute Within an Independent Module

If you need to call a function in the current independent module using Execute, you can compose the name using the **GetIndependentModuleName** function. For example, outside of an independent module the commands would be:

```
String cmd = "WS_UpdateWaveSelectorWidget(\"Panel0\", \"selectorWidgetName\")"
Execute cmd
```

But inside an independent module the commands are:

```
#pragma IndependentModule=MyIM
String cmd="WS_UpdateWaveSelectorWidget(\"Panel0\", \"selectorWidgetName\")"
cmd = GetIndependentModuleName() + "#" + cmd    // Make qualified name
Execute cmd
```

You could change the command string to:

```
cmd = "MyIM#" + cmd
```

but using GetIndependentModuleName allows you to disable the IndependentModule pragma by commenting it out and have the code still work, which can be useful during development. With the pragma commented out you are running in ProcGlobal context and GetIndependentModuleName returns "ProcGlobal".

## Independent Modules and Dependencies

**GetIndependentModuleName** is also useful for defining dependencies using functions in the current independent module. Dependencies are evaluated in the global procedure context (ProcGlobal). In order for