```
End
...
PopupMenu pop0 value=#"MyPopMenuList()"      // Note the quotation marks
```

But inside an independent module you need this:

```
#pragma IndependentModule=MyIM
Function/S MyPopMenuList()
   return "Item 1;Item2;"
End
...
String cmd= GetIndependentModuleName()+"#MyPopMenuList()"
PopupMenu pop0 value=#cmd                    // No enclosing quotation marks
```

GetIndependentModuleName returns the name of the independent module to which the currently-running function belongs or "ProcGlobal" if the currently-running function is not part of an independent module.

You could change the command string to:

```
PopupMenu pop0 value=#"MyIM#MyPopMenuList()"
```

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".

When the user clicks the popup menu, Igor generates the menu items by evaluating the text specified by the PopupMenu value keyword as an Igor expression. The expression (`"MyIM#MyPopMenuList()"` in this case) is evaluated in the ProcGlobal context. In order for Igor to find the function in the independent module, it must be public (non-static), except as describe under **Regular Modules Within Independent Modules** on page IV-242, and you must use a qualified name.

Note that `#cmd` is not the same as `#"cmd"`. The `#cmd` form was introduced with Igor Pro 6. The string variable cmd is evaluated when PopupMenu runs which occurs in the context of the independent module. The contents of cmd (`"MyIM#MyPopMenuList()"` in this case) are stored in the popup menu's internal data structure. When the popup menu is clicked, Igor evaluates the stored text as an Igor expression in the ProcGlobal context. This causes the function `MyIM#MyPopMenuList` to run.

With the older `#"cmd"` syntax, the stored text is evaluated only when the popup menu is clicked, not when the PopupMenu operation runs, and this evaluation occurs in the ProcGlobal context. It is too late to capture the independent module in which the text should be evaluated.

Regular Modules Within Independent Modules

It is usually not necessary but you can create a regular module within an independent module. For example:

```
#pragma IndependentModule = IndependentModuleA
#pragma ModuleName = RegularModuleA
Function Test()
   Print "Test in RegularModuleA within IndependentModuleA"
End
```

Here RegularModuleA is a regular module within IndependentModuleA.

To call the function Test from outside of the independent module you must qualify the call like this:

```
IndependentModuleA#RegularModuleA#Test()
```

This illustrates that the independent module establishes its own namespace (IndependentModuleA) which can host one level of sub-namespace (RegularModuleA). By contrast, a regular module creates a namespace within the global namespace (called ProcGlobal) and can not host additional sub-namespaces.