Chapter
IV-9

# Dependencies

# Dependency Formulas

Igor Pro supports "dependent objects". A dependent object can be a wave, a global numeric variable or a global string variable that has been linked to an expression. The expression to which an object is linked is called the object's "dependency formula" or "formula" for short.

The value of a dependent object is updated whenever any other global object involved in the formula is modified, even if its value stays the same. We say that the dependent object *depends* on these other global objects *through* the formula.

You might expect that an assignment such as:

```
Variable var0 = 1
Make wave0
wave0 = sin(var0*x/16)
```

meant that wave0 would be updated whenever var0 changed. It doesn't. Values are computed for wave0 only once, and the relationship between wave0 and var0 is forgotten.

However, if the equal sign in the above assignment is replaced by a colon followed by an equal sign:

```
wave0 := sin(var0*x/16)
```

then Igor *does* create just such a dependency. Now whenever var0 changes, Igor reevaluates the assignment statement and updates wave0. In this example, wave0 is a dependent object. It depends on var0, and "sin(var0*x/16)" is wave0's dependency formula.

You can also establish a dependency using the SetFormula operation, like this:

```
SetFormula wave0, "sin(var0*x/16)"
```

Wave1 depends on var0 because var0 is a changeable variable. Wave1 *also* depends on the function x which returns the X scaling of the destination wave (wave0). When the X scaling of wave0 changes, the values that the x function returns change, so this dependency assignment is reevaluated. The remaining terms (sin and 16) are not changeable, so wave0 does not depend on them.

From the command line, you can use either a dependency assignment statement or SetFormula to establish a dependency. In a user-defined function, you must use SetFormula.

Like other assignment statements, the data folder context for the right-hand side is that of the destination object. Therefore, in the following example, wave2 and var1 must be in the data folder named foo, var2 must be in root, and var3 must be in root:bar.

```
root:foo:wave0 := wave2*var1 + ::var2 + root:bar:var3
```

Data Folders are described in Chapter II-8, **Data Folders**.

A dependency assignment is often used in conjunction with SetVariable controls and ValDisplay controls.

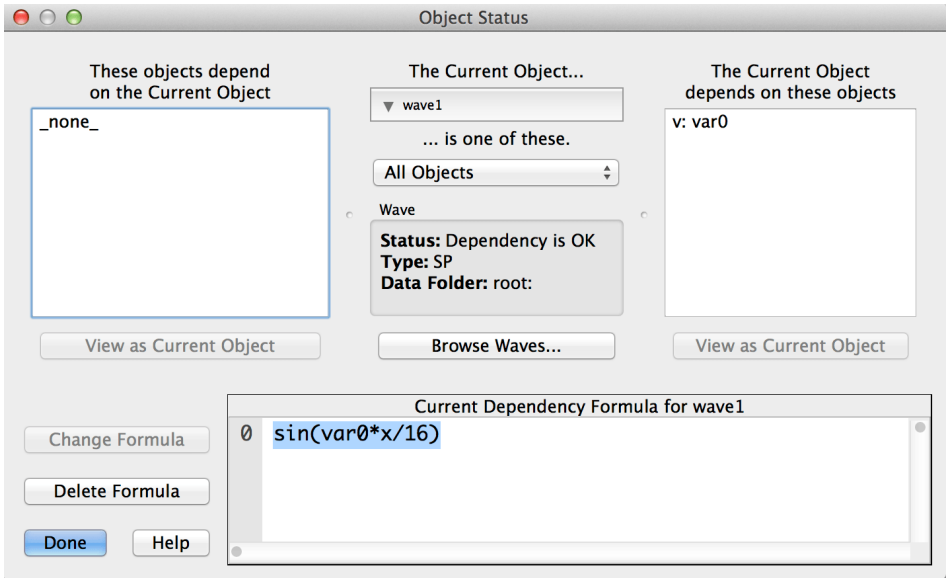
Here's a simple example. Execute these commands on the command line:

```
Variable/G var0 = 1
Make/O wave0:=sin(var0*x/16)
Display /W=(4,53,399,261) wave0
ControlBar 23
SetVariable setvar0,size={60,15},value=var0
```

Click the SetVariable control's up and down arrows to adjust var0 and observe that wave0 is automatically updated.

# Dependencies and the Object Status Dialog

You can use the Object Status dialog, which you can access via the the Misc menu, to check dependencies. After executing the commands in the preceding section, the dialog looks like this:




The Status area indicates any dependency status:

- "No dependency" means that the current object does not depend on anything.
- "Dependency is OK" means that the dependency formula successfully updated the current object.
- "Update failed" means that the dependency formula used to compute the current object's value failed.

An update may fail because there is a syntax error in the formula or one of the objects referenced in the formula does not exist or has been renamed. If the formula includes a call to a user-defined function then the update will fail if the function does not exist or if procedures are not compiled.

If an update fails, then the objects that depend on that update are broken. See **Broken Dependent Objects** on page IV-217 for details.

You can create a new dependency formula with the New Formula button, which appears only if the current object is not the target of a dependency formula.

You can delete a dependency formula using the Delete Formula button.

You can change an existing dependency formula by typing in the Dependency Formula window, and clicking the Change Formula button.

For further details on the Object Status dialog, click the Help button in the dialog.

# Numeric and String Variable Dependencies

Dependencies can also be created for global user-defined numeric and string variables. You can not create a dependency that uses a local variable on either side of the dependency assignment statement.

Here is a user-defined function that creates a dependency. The global variable `recalculateThis` is made to depend on the global variable `dependsOnThis`:

```
Function TestRecalculation()
   Variable/G recalculateThis
   Variable/G dependsOnThis = 1

   // Create dependency on global variable
   SetFormula recalculateThis, "dependsOnThis"

   Print recalculateThis      // Prints original value

   dependsOnThis = 2          // Changes something recalculateThis
```

```
   DoUpdate                    // Make Igor recalculate formulae

   Print recalculateThis       // Prints updated value
End
```

Running this function prints the following to the history area:

```
•TestRecalculation()
   1
   2
```

The call to DoUpdate is needed because Igor recalculates dependency formulas only when no user-defined functions are running or when DoUpdate is called.

This function uses SetFormula to create the dependency because the := operator is not allowed in user-defined functions.

# Wave Dependencies

The assignment statement:

*dependentWaveName* := *formula*

creates a dependency and links the dependency formula to the dependent wave. Whenever any change is made to any object in the formula, the contents of the dependent wave are updated.

The command

SetFormula *dependentWaveName*, "*formula*"

establishes the same dependency.

From the command line, you can use either a dependency assignment statement or SetFormula to establish a dependency. In a user-defined function, you must use SetFormula.

# Cascading Dependencies

"Cascading dependencies" refers to the situation that arises when an object depends on a second object, which in turn depends on a third object, etc. When an object changes, all objects that directly depend on that object are updated, *and* objects that depend directly on those updated objects are updated until no more updates are needed.

You can use the Object Status dialog to investigate cascading dependencies.

# Deleting a Dependency

A dependency is deleted when the dependent object is assigned a value using the = operator:

```
recalculateThis := dependsOnThis    // Creates a dependency
recalculateThis = 0                 // Deletes the dependency
```

This method of deleting a dependency does not work in user-defined functions. You must use the SetFormula operation.

For example:

```
Execute "recalculateThis = 0"
```

will delete the dependency even in a user-defined function.

You can also delete this dependency using the SetFormula operation.

```
SetFormula recalculateThis, ""
```

Wave dependencies are also deleted by operations that overwrite the values of their wave parameters. Some of these operations are:

```
FFT      Convolve       Correlate   Smooth   GraphWaveEdit
Hanning  Differentiate  Integrate   UnWrap
```

Dependencies can also be deleted using the Object Status dialog.

# Broken Dependent Objects

Igor compiles the text of a dependency formula to low-level code and stores both the original text and the low-level code with the dependent object. At various times, Igor may need to recompile the dependency formula text. At that time, a compilation error will occur if:

- The dependency formula contains an error
- An object used in the dependency formula has been deleted, renamed, or moved
- The dependency formula references a user-defined function that is missing
- The dependency formula references a user-defined function and procedures are not compiled

When this happens, the dependent object will no longer update but will retain its last value. We call such an object "broken".

To inspect broken objects, invoke the Object Status dialog. Choose Broken Objects from the pop-up menu above the status area. If there are any broken objects, they will appear in the Current Object pop-up. Select an object from the Current Object pop-up to inspect it.

# When Dependencies are Updated

Dependency updates take place at the same time that graphs are updated. This happens after each line in a macro is executed, or when DoUpdate is called from a macro or user function, or continuously if a macro or function is not running.

Dependency formulas used as input to the SetBackground and ValDisplay operations, and in some other contexts, can alternately be specified as a literal string of characters using the following syntax:

```
#"text_of_the_dependency_expression"
```

Note that what follows the # char must be a literal string — not a string expression.

This sets the dependency formula *without* compiling it or checking it for validity. If you need to set the dependency formula of an object to something that is not currently valid but will be in the future, then use this alternate method.

# Programming with Dependencies

You cannot use := to create dependencies in user-defined functions. Instead you must use the **SetFormula** operation (see page V-723).

```
Function TestFunc()
   Variable/G varNum=-666
   Make wave0
   SetFormula wave0, "varNum"    // Equivalent to wave0 := varNum
End
```

# Using Operations in Dependency Formulas

The dependency formula must be a single expression — and you can not use operations, such as FFT's, or other command type operations. However, you can invoke user-defined functions which invoke operations. For example:

```
Function MakeDependencyUsingOperation()
   Make/O/N=128 data = p         // A ramp from 0 to 127
   Variable/G power

   SetFormula power, "RMS(data)" // Dependency on function and wave
   Print power

   data = p * 2                  // Changes something power depends
   DoUpdate                      // Make Igor recalc formulae
   Print power
EndMacro

Function RMS(w)
   Wave w

   WaveStats/Q w           // An operation! One output is V_rms
   return V_rms
End
```

When `MakeDependencyUsingOperation` is executed, it prints the following in the history area:

```
•MakeDependencyUsingOperation()
   73.4677
   146.935
```

# Dependency Caveats

The extensive use of dependencies can create a confusing tangle that can be difficult to manage. Although you can use the Object Status dialog to explore the dependency hierarchy, you can still become very confused very quickly, especially when the dependencies are highly cascaded. You should use dependencies only where they are needed. Use conventional assignments for the majority of your calculations.

Dependency formulas are generally not recalculated when a user-defined function is running unless you explicitly call the DoUpdate operation. However, they can run at other unpredictable times so you should not make any assumptions as to the timing or the current data folder when they run.

The text of the dependency formula that is saved for a dependent object is the original literal text. The dependency formula needs to be recompiled from time to time, for example when procedures are compiled. Therefore, any objects used in the formula must persist until the formula is deleted.

We recommend that you never use $ expressions in a dependency formula.