

The `kCCE_frame` event is sent just before drawing one of the *pict* frames, as set by the picture parameter. On input, the `curFrame` field is set to 0 (normal or mouse down outside button), to 1 (mouse down in button), or to 2 (disable). You may modify `curFrame` as desired but your value will be clipped to a valid value.

When you specify a *pict* with the picture parameter, you will get a `kCCE_drawOSBM` event when that *pict* is drawn into an offscreen bitmap. Once it is created, all updates use the offscreen bitmap until you specify a new picture parameter. Thus the custom drawing done at this event is static, unlike drawing done during the `kCCE_draw` event, which can be different each time the control is drawn. Because the *pict* can contain multiple side-by-side frames, the width of the offscreen bitmap is the width derived from the `ctrlRect` field multiplied by the number of frames.

Because the action function is called in the middle of various control events, it must not kill the control or the window. Doing so will almost certainly cause a crash.

CustomControl Action Procedure

The action procedure for a CustomControl control can take a predefined structure `WMCustomControlAction` as a parameter to the function:

```
Function ActionProcName(s)
    STRUCT WMCustomControlAction& s
    ...
    return 0
End
```

See **WMCustomControlAction** for details on the `WMCustomControlAction` structure.

Although the return value is not currently used, action procedures should always return zero.

The action procedure may have drawing commands such as **SetDrawEnv**, **DrawRect**, etc. These commands include a `/W` flag that directs their action to a particular window or subwindow. However, in a CustomControl action procedure, the `/W` flag is ignored and all drawing commands are directed to the window or subwindow containing the control.

Because the action procedure runs during a drawing operation that cannot be interrupted without crashing Igor, the debugger cannot be invoked while it is running. Consequently breakpoints set in the function are ignored. Use **Debugging With Print Statements** on page IV-212 instead.

Examples

See **Creating Custom Controls** on page III-424 for some examples of custom controls.

For a demonstration of custom controls, choose File→Example Experiments→Feature Demos 2→Custom Control Demo.

See Also

Chapter III-14, **Controls and Control Panels**, for details about control panels and controls.

Control Panel Units on page III-444 for a discussion of the units used for controls.

The **ControlInfo** operation for information about the control.

The **GetUserData** function for retrieving named user data.

Proc Pictures on page IV-56.

The **TextBox**, **DrawPoly** and **DefaultGUIControls** operations.

CWT

CWT [*flags*] *srcWave*

The CWT operation computes the continuous wavelet transform (CWT) of a 1D real-valued input wave (*srcWave*). The input can be of any numeric type. The computed CWT is stored in the wave `M_CWT` in the current data folder. `M_CWT` is a double precision 2D wave which, depending on your choice of mother wavelet and output format, may also be complex. The dimensionality of `M_CWT` is determined by the specifications of offsets and scales. The operation sets the variable `V_flag` to zero if successful or to a nonzero number if it fails for any reason.

Flags

<i>/ENDM=method</i>	<p>Selects the method used to handle the two ends of the data array with direct integration (/M=1).</p> <p><i>method=0:</i> Padded on both sides by zeros.</p> <p><i>method=1:</i> Reflected at both the start and end.</p> <p><i>method=2:</i> Entered with cyclical repetition.</p>
<i>/FSCL</i>	<p>Use correction factor to the wave scaling of the second dimension of the output wave so that the numbers are more closely related to Fourier wavelength. See References for more information on the calculation of these correction factors. This flag does not affect the output from the Haar wavelet.</p>
<i>/M=method</i>	<p>Specifies the CWT computation method.</p> <p><i>method=0:</i> Fast method uses FFT (default).</p> <p><i>method=1:</i> Slower method using direct integration.</p> <p>You should mostly use the more efficient FFT method. The direct method should be reserved to situations where the FFT is not producing optimal results. Theoretically, when the FFT method fails, the direct method should also be fairly inaccurate, e.g., in the case of undersampled signal. The main advantage in the direct method is that you can use it to investigate edge effects.</p>
<i>/OSCW</i>	<p>CWT creates an output scaling wave named W_CWTScaling that can be used as the Y wave in an image plot. For example:</p> <pre>Display; AppendImage M_CWT vs {*,W_CWTScaling}</pre> <p>If you omit <i>/OSCW</i>, CWT creates W_CWTScaling only if you include <i>/SMP2=4</i>.</p> <p><i>/OSCW</i> was added in Igor Pro 9.00.</p>
<i>/OUT=format</i>	<p>Sets the format of the output wave M_CWT:</p> <p><i>format=1:</i> Complex.</p> <p><i>format=2:</i> Real valued.</p> <p><i>format=4:</i> Real and contains the magnitude.</p> <p>Depending on the method of calculation and the choice of mother wavelet, the "native" output of the transform may be real or complex. You can force the output to have a desired format using this flag.</p>
<i>/Q</i>	<p>Quiet mode; no results printed to the history.</p>
<i>/R1={startOffset, delta1, numOffsets}</i>	<p>Specifies offsets for the CWT. Offsets are the first dimension in a CWT. Normally you will calculate the CWT for the full range of offsets implied by <i>srcWave</i> so you will not need to use this flag. However, when using the slow method, this flag restricts the output range of offsets and save some computation time. <i>startOffset</i> (integer) is the point number of the first offset in <i>srcWave</i>. <i>delta1</i> is the interval between two consecutive CWT offsets. It is expressed in terms of the number <i>srcWave</i> points. <i>numOffsets</i> is the number of offsets for which the CWT is computed.</p> <p>By default <i>startOffset=0</i>, <i>delta1=1</i>, and <i>numOffsets</i> is the number of points in <i>srcWave</i>. If you want to specify just the <i>startOffset</i> and <i>delta1</i>, you can set <i>numOffsets=0</i> to use the same number of points as the source wave.</p>

/R2={startScale, scaleStepSize, numScales}

Specifies the range of scales for the CWT is computation. Scales are the second dimension in the output wave. Note however that there are limitations on the minimum and maximum scales having to do with the sampling of your data. Because there is a rough correspondence between a Fourier spatial frequency and CWT scale it should be understood that there is also a maximum theoretical scale. This is obvious if you compute the CWT using an FFT but it also applies to the slow method. If you specify a range outside the allowed limits, the corresponding CWT values are set to NaN.

Use NaN if you want to use the default value for any parameter.

The default value for *startScale* is determined by sampling of the source wave and the wavelet parameter or order.

At a minimum you must specify either *scaleStepSize* or *numScales*.

/SMP1=offsetMode Determines computation of consecutive offsets. Currently supporting only *offsetMode=1* for linear, user-provided partial offset limits (see */R1* flag):
 $val1 = startOffset + numOffsets * delta1.$

/SMP2=scaleMode Determines computation of consecutive scales. *scaleMode* is 1 by default if you specify the */R2* flag.

format=1: Linear:
 $theScale = startScale + index * scaleStepSize$
format=2: Auto:
 $theScale = numScales * startScale / (index + 1)$
format=4: Power of 2 scaling interval:
 $theScale = startScale * 2.^{(index * scaleStepSize)}$

When using *scaleMode=4* the operation saves the consecutive scale values in the wave *W_CWTScaling*. Note also that if you use *scaleMode=4* without specifying a corresponding */R2* flag, the default *scaleStepSize* of 1 and 64 scale values gives rise to scale values that quickly exceed the allowed limits.

(See */R2* flag for details about the different parameters used in the equations above.)

/SUBM Subtracts the mean of the input before performing the transform. */SUBM* was added in Igor Pro 9.00.

/SW2=sWave Provides specific scale values at which the transform is evaluated. Use instead of */R2* flag. It is your responsibility to make sure that the entries in the wave are appropriate for the sampling density of *srcWave*.

/WBI1={Wavelet [, order]}

Specifies the built-in wavelet (mother) function. *Wavelet* is the name of a wavelet function: Morlet (default), MorletC (complex), Haar, MexHat, DOG, and Paul.

Morlet:
$$\Psi_0(x) = \frac{1}{\pi^{1/4}} \cos(\omega x) \exp\left(-\frac{x^2}{2}\right).$$

By default, $\omega=5$. Use the /WPR1 flag to specify other values for ω .

MorletC:
$$\Psi_0(x) = \frac{1}{\pi^{1/4}} \exp(i\omega x) \exp\left(-\frac{x^2}{2}\right).$$

By default, $\omega=5$. Use the /WPR1 flag to specify other values for ω .

Haar:
$$\Psi_0(x) = \begin{cases} 1 & 0 \leq x < 0.5 \\ -1 & 0.5 \leq x < 1 \end{cases}.$$

DOG:
$$\Psi_0(x) = \frac{(-1)^{m+1}}{\sqrt{\Gamma\left(m + \frac{1}{2}\right)}} \frac{d^m}{dx^m} \left(\exp\left(-\frac{x^2}{2}\right) \right).$$

MexHat: Special case of DOG with $m=2$.

Paul:
$$\Psi_0(m, x) = \frac{2^m i^m m!}{\sqrt{\pi (2m)!}} (1 - ix)^{-(m+1)}.$$

order applies to DOG and Paul wavelets only and specifies m , the particular member of the wavelet family.

The default wavelet is the Morlet.

/WPR1={param1} *param1* is a wavelet-specific parameter for the wavelet function selected by /WBI1. For example, use /WPR1={6} to change the Morlet frequency from the default (5).

/Z No error reporting. If an error occurs, sets V_flag to -1 but does not halt function execution.

Details

The CWT can be computed directly from its defining integral or by taking advantage of the fact that the integral represents a convolution which in turn can be calculated efficiently using the fast Fourier transform (FFT).

When using the FFT method one encounters the typical sampling problems and edge effects. Edge effects are also evident when using the slow method but they only significant in high scales.

From sampling considerations it can be shown that the maximum frequency of a discrete input signal is $1/2dt$ where dt is the time interval between two samples. It follows that the smallest CWT scale is $2dt$ and the largest scale is Ndt where N is the total number of samples in the input wave.

The transform in M_CWT is saved with the wave scaling. *startOffset* and *delta1* are used for the X-scaling. Both *startOffset* and *delta1* are either specified by the /R1 flag or copied from *srcWave*. The Y-scaling of M_CWT depends on your choice of /SMP2. If the CWT scaling is linear then the wave scaling is based on *startScale* and *scaleStepSize*. If you are using power of 2 scaling interval then the Y wave scaling of M_CWT has a *start=0* and *delta=1* and the wave W_CWTScaling contains the actual scale values for each column of M_CWT. Note that W_CWTScaling has one extra data point to make it suitable for display using an operation like:

AppendImage M_CWT vs {*, W_CWTScaling}