Normally, a pop-up menu rebuilds only when the user clicks on it. If you set up a pop-up menu so that its contents depend on a global string variable, on a user-defined string function or on an Igor function (e.g., **WaveList**), you may want to force the pop-up menu to be updated at your command.

Usually, a ValDisplay control displays the value of a global variable or of an expression involving a global variable. If the global variable changes, the ValDisplay will automatically update. However, you can create a ValDisplay that displays a value that does not depend on a global variable. For example, it might display the result of an external function. In a case like this, the ValDisplay will not automatically update. You can update it by calling ControlUpdate.

When a SetVariable control is being edited, the text the user types isn't "accepted" (or processed) until the user presses Return or Enter. ControlUpdate effectively causes the named control to act as though the user has pressed one of those keys. If /A is specified, the currently active SetVariable control (if any) is affected this way. The motivation here is that the user may have typed a new value without having yet pressed return, and then may click a button in a different panel which runs a routine that uses the SetVariable value as input. The user expected the typed value to have been accepted but the variable has not yet been set. Calling ControlUpdate/A on the first panel will read the typed value in the variable, avoiding a discrepancy between the visible value of the SetVariable control and the actual value of the variable.

Examples

```
NewPanel;DoWindow/C PanelX
String/G popupList="First;Second;Third"
PopupMenu oneOfThree value=popupList        // popup shows "First"
popupList="1;2;3"                           // popup is unchanged
ControlUpdate/W=PanelX oneOfThree           // popup shows "1"
```

See Also

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

ConvertGlobalStringTextEncoding

ConvertGlobalStringTextEncoding [flags] originalTextEncoding, newTextEncoding, [string , string, ...]

The ConvertGlobalStringTextEncoding operation converts the contents of the specified global string variables or the contents of all global string variables in the data folder specified with the /DF flag from one text encoding to another.

The ConvertGlobalStringTextEncoding operation was added in Igor Pro 7.00.

By default, the contents of global string variables containing control characters (codes less than 32) other than carriage return (13), linefeed (10), and tab (9) are not converted. To convert the contents of global string variables containing control characters, you must include the /SKIP=0 flag.

Also by default, in Igor Pro 9.00 or later, when converting to UTF-8, strings that are already valid as UTF-8 are skipped.

In Igor Pro 7 or later, the contents of all global string variables are assumed to be encoded as UTF-8. You should convert the text encoding of a global string variable if, for example, you have global string variables in experiments created prior to Igor7 and they contain non-ASCII characters. For those strings to be displayed and interpreted correctly, their contents need to be converted to UTF-8.

Most users will have no need to worry about the text encoding of Igor global string variables since most global string variables do not contain non-ASCII text. You should not use this operation unless you have a thorough understanding of text encoding issues or are instructed to use it by someone who has a thorough understanding.

See **String Variable Text Encodings** on page III-478 for essential background information.

ConvertGlobalStringTextEncoding can work on a list of specific global string variables or on all of the global string variables in a data folder (/DF flag). When working on a data folder, it can work on just the data folder itself or recursively on sub-data folders as well.

Conversion does not change the characters that make up text - it merely changes the numeric codes used to represent those characters.

Parameters

originalTextEncoding specifies the original (current) text encoding used by the specified global string variables. See **Text Encoding Names and Codes** on page III-490 for a list of codes. It will typically be 2 (MacRoman), 3 (Windows-1252) or 4 (Shift JIS), depending on the system on which the string variables were created.

newTextEncoding specifies the output text encoding. See **Text Encoding Names and Codes** on page III-490 for a list of codes. It will typically be 1 which stands for UTF-8.

string , *string* , ... is a list of targeted global string variables. Only the name of string variables, not a path specification plus the name, is allowed. Therefore, the string variables must be in the current data folder. The list is optional and must be omitted if you use the /DF flag. Using both the /DF flag and a list of string variables is treated as an error. Use of local string variables in this list is also an error. Use **ConvertTextEncoding** to convert local string variables.

Flags

/CONV={*errorMode* [, *diagnosticsFlags*]}

errorMode determines how ConvertGlobalStringTextEncoding behaves if the conversion can not be done because the text can not be mapped to the specified text encoding. This occurs if the string variable's original text is not valid in the specified *originalTextEncoding* or if it contains characters that can not be represented in the specified *newTextEncoding*.

errorMode takes one of these values:

1: Generate error. SetWaveTextEncoding returns an error to Igor.

2: Use a substitute character for any unmappable characters. The substitute character for Unicode is the Unicode replacement character, U+FFFD. For most non-Unicode text encodings it is either control-Z or a question mark.

3: Skip unmappable input characters. Any unmappable characters will be missing in the output.

4: Use escape sequences representing any unmappable characters or invalid source text.

If the source text is valid in the source text encoding but can not be represented in the destination text encoding, unmappable characters are replaced with \uXXXX where XXXX specifies the UTF-16 code point of the unmappable character in hexadecimal.

If the conversion can not be done because the source text is not valid in the source text encoding, invalid bytes are replaced with \xXX where XX specifies the value of the invalid byte in hexadecimal.

diagnosticsFlags is an optional bitwise parameter defined as follows:

Bit 0: Emit diagnostic message if text conversion succeeds.

Bit 1: Emit diagnostic message if text conversion fails.

Bit 2: Emit diagnostic message if text conversion is skipped.

Bit 3: Emit summary diagnostic message.

All other bits are reserved for future use.

See **Setting Bit Parameters** on page IV-12 for details about bit settings.

diagnosticsFlags defaults to 6 (bits 1 and 2 set) if the /DF flag is not present and to 14 (bits 1, 2 and 3 set) if the /DF flag is present.

ConvertGlobalStringTextEncoding skips text conversion if the global string variable's contents are detected as binary and you omit /SKIP=0.

/DF={*dfr*, *recurse*, *excludedDFR*}

dfr is a reference to a data folder. ConvertGlobalStringTextEncoding operates on all global string variables in the specified data folder. If *dfr* is null ($"") ConvertGlobalStringTextEncoding acts as if /DF was omitted.

If you use the /DF flag, you must omit the optional string variable list.

If *recurse* is 1, ConvertGlobalStringTextEncoding works recursively on all sub-data folders. Otherwise it affects only the data folder referenced by *dfr*.

excludedDFR is an optional reference to a data folder to be skipped by ConvertGlobalStringTextEncoding. For example, this command converts the string data for all global string variables in all data folders except for root:Packages and its sub-data folders:

```
ConvertGlobalStringTextEncoding /DF={root:,1,root:Packages} 4, 1
```

If *excludedDFR* is null ($"") ConvertGlobalStringTextEncoding acts as if *excludedDFR* was omitted and no data folders are excluded.

/SKIP=*skip*

Skips conversion of a string variable as follows:

skip=0:	Do not skip conversion.
skip=1:	Skip conversion of string variables containing binary data.
skip=2:	When converting to UTF-8, skip conversion of strings that are already valid as UTF-8. This includes strings that contain only ASCII characters.
skip=3:	Skips both strings containing binary data and, when converting to UTF-8, strings that are already valid as UTF-8. This is the default behavior if /SKIP is omitted.

/SKIP=2 and /SKIP=3 require Igor Pro 9.00 or later.

/Z[=*z*]

Prevents procedure execution from aborting if ConvertGlobalStringTextEncoding generates an error. Use /Z or the equivalent, /Z=1, if you want to handle errors in your procedures rather than having execution abort.

/Z does not suppress invalid parameter errors. It suppresses only errors in doing text encoding reinterpretation or conversion.

Details

For general background information on text encodings, see **Text Encodings** on page III-459.

For background information on string variable text encodings, see **String Variable Text Encodings** on page III-478.

Using ConvertGlobalStringTextEncoding

ConvertGlobalStringTextEncoding is used to change the numeric codes representing the text - i.e., to convert the content to a different text encoding. Its main use is to convert Igor Pro 6 global string variables from whatever text encoding they use to UTF-8. UTF-8 is a form of Unicode which is more modern but is not backward compatible with Igor Pro 6. Igor Pro 7 and later assume that string variables are encoded as UTF-8.

For example, if you have global string variables from Igor Pro 6 that are encoded in Japanese (Shift JIS), you should convert them to UTF-8. Otherwise you will get errors or garbled text when you print or display the strings. This also applies to western text containing non-ASCII characters encoded as MacRoman or Windows-1252.

If you have an Igor6 experiment containing non-ASCII string variables encoded as, for example, MacRoman, and you add some non-ASCII string variables in Igor7 or later, you now have a mix of MacRoman and UTF-8 non-ASCII string variables. In Igor9 or later, when you are converting to UTF-8, by default ConvertGlobalStringTextEncoding skips conversion of strings that are already valid as UTF-8. This allows you to convert the MacRoman string variables to UTF-8 without messing up the string variables that are already UTF-8.