ITK 4.0 Changes

System requirements
ITK 4.0 requires 4D v14 or above.

Unicode text
ITK 4.0 is a native post-4D v11 plugin, which means that it can work with Unicode text passed to and from 4D. This change affects all ITK commands that deal with text. In previous versions the maximum length of text that could be passed to or from ITK was 32000 characters. Now the maximum length than can be **passed** to and from 4D is 2GB, however, if the text is going to be converted to UTF-8 within an ITK command (by using the kITKConvertToUTF8 filter), the effective limit is 682MB.

Filters
The current set of filters is as follows:

Value	Description
1	convert Mac text to ISO 8859-1
2	convert ISO 8859-1 to Mac text
4	convert CR to CR/LF
8	convert CR/LF to CR
16	convert Mac text to HTML
32	convert HTML to Mac text
64	convert Mac text to HTML – do not convert HTML tags ('<', '>' or '"')
128	convert Mac text to Base64
256	convert Base64 to Mac text
512	convert Mac text to Quoted-Printable
1024	convert Quoted-Printable to Mac text

Some of these filters are character encoding conversions, and some are text transformations. In ITK 4.0, the encoding conversions apply generically to mean convert from or to a given character encoding, thus there are “from” and “to” conversions. To make it easier to specify encoding conversions in your code, there are now named constants for each conversion, and some additional conversions have been added to support other common encodings.

Here are the new filters with their associated constants:

Value	Description	Constant
1	convert to ISO 8859-1	kITKConvertToISO_8859_1
2	convert from ISO 8859-1	kITKConvertFromISO_8859_1
4	convert CR to CR/LF	kITKConvertToCRLF
8	convert CR/LF to CR	kITKConvertToCR
16	convert to HTML	kITKConvertToHTML
32	convert from HTML	kITKConvertFromHTML
64	convert to HTML (no tags)	kITKConvertToHTMLNoTags
128	convert to Base64	kITKConvertToBase64
256	convert from Base64	kITKConvertFromBase64
512	convert to Quoted-Printable	kITKConvertToQuoted
1024	convert from Quoted-Printable	kITKConvertFromQuoted
2048	convert to UTF-8	kITKConvertToUTF8
4096	convert from UTF-8	kITKConvertFromUTF8
8192	convert to Mac Roman	kITKConvertToMacRoman
16384	convert from Mac Roman	kITKConvertFromMacRoman

In general, when Unicode text is sent over the network, it is always converted from Unicode to another encoding. If the command has a *filter* parameter and no “to” conversion is specified in the filter, it defaults to kITKConvertToUTF8 to ensure all Unicode characters can be represented. When receiving text over the network, it is always converted to Unicode. If the command has a *filter* parameter and no “from” conversion is specified, it defaults to kITKConvertFromUTF8.

Paths

All commands that take a file path (except ITK_OpenFile) can now take a Posix-style path on OS X. For example, instead of “System:Users:homer:Documents:marge.jpg”, you can use “/Users/homer/Documents/marge.jpg”. If a path contains “:”, it is assumed to be an HFS path.

Lost connections
If a client connection is terminated by the server, the first ITK_TCPSend after that will not return an error. But the second send will return kStreamStatusInvalid (-1) and the stream will be released. To determine if the stream was released (vs. some other send error), check if ITK_TCPStatus(streamRef)=kStreamStatusInvalid.

ITK_TCPSend
A new named constant has been added for use with the *flushFlag* parameter:

kITKBuffer = 1

If no “to” encoding conversion is specified in the *filter* parameter, it defaults to UTF-8. If a “from” encoding conversion is passed in the *filter* parameter, -7 (invalid conversion) is returned.

ITK_TCPRcv
New named constants have been added for use with the *options* parameter:

kITKKeepEndString = 2
kITKSearchNewData = 4

If no “from” encoding conversion is specified in the *filter* parameter, it defaults to UTF-8. If a “to” encoding conversion is passed in the *filter* parameter, -7 (invalid conversion) is returned.

ITK_TCPUnRcv
ITK_TCPUnRcv now has an additional longint parameter, *filterFlag*, which determines what encoding is used in the unreceive buffer. If not specified, it defaults to kITKConvertFromUTF8. It should always be the same as the encoding used in ITK_TCPRcv.

There are two new named constants for use with the *options* parameter:

kITKAppendUnreceiveData = 0
kITKInsertUnreceiveData = 1

ITK_TCPSendFile
When using ITK_TCPSendFile, there are two encoding conversions: “from” conversion (when the file is read from disk) and “to” conversion (when the file is sent over the network). If no conversion is specified in the *filter* parameter (e.g. kITKConvertFromISO_8859_1), the file is considered binary data and is sent as is.

Otherwise, if you specify only one conversion (“from” or “to”), the other conversion defaults to the one you specify.

For example, if you have a UTF-8 file that you want to send as ISO-8859-1, you would need to add kITKConvertFromUTF8+kITKConvertToISO_8859_1 to the *filter* parameter.

Important: If you are sending a UTF-8 file, you should always add kITKConvertFromUTF8 to the filter, even if no encoding conversion is being done. This enables ITK to do special processing that helps to ensure valid UTF-8 is received.

Warning: When converting to or from UTF-8 (by using an encoding conversion such as kITKConvertFromISO_8859_1+kITKConvertToUTF8) and not using any transformation filters (such as kITKConvertToCR), it is possible that the length of the text will change before being sent. Therefore you cannot reliably use **Get document size** to specify the number of bytes being sent. If the receiver needs to know exactly how many bytes to receive, then you must either:

1. Convert the document text to a blob in the “to” encoding and use **ITK_TCPSendBlob** with no encoding conversion instead of **ITK_TCPSendFile**. This is the preferred method, as it avoids converting the text twice.
2. Convert the document text to a blob in the “to” encoding and use the blob’s size as the number of bytes to be received.

Here is an example of method #1:

```
// Instead of using kITKConvertFromUTF8+kITKConvertToISO_8859_1
DOCUMENT TO BLOB(Document;$blob)
$text:=Convert to text($blob;"utf-8")
CONVERT FROM TEXT($text;"iso-8859-1";$blob)

// Send a header to the receiver that indicates the length of the data being sent
sendHeader (BLOB size($blob))

$err:=ITK_SendBlob ($stream;$blob)
```

If an error occurs, a non-zero value is returned.

ITK_TCPRecvFile
When using ITK_TCPRecvFile, there are two encoding conversions: “from” conversion (when the file is received over the network) and “to” conversion (when the file is written to disk).

If no conversion is specified in the *filter* parameter (e.g. kITKConvertFromISO_8859_1), the received data is considered binary data and is saved as is. Otherwise, if you specify only one conversion (“from” or “to”), the other conversion defaults to the one you specify.

For example, if you receive data as UTF-8 and want to save it as ISO-8859-1, you would need to add kITKConvertFromUTF8+kITKConvertToISO_8859_1 to the filter.

If an error occurs, a non-zero value is returned.

ITK_TCPSendBlob

When using ITK_TCPSendBlob and the blob contains text, there are two encoding conversions: “from” conversion (the encoding of the text in the blob) and “to” conversion (the encoding in which to send the text over the network). If no conversion is specified in the *filter* parameter (e.g. kITKConvertFromISO_8859_1), the blob data is considered binary data and is sent as is. Otherwise, if you specify only one conversion (“from” or “to”), the other conversion defaults to the one you specify.

Important: If you are sending UTF-8 text in a blob, you should always add kITKConvertFromUTF8 to the filter, even if no encoding conversion is being done. This enables ITK to do special processing that helps to ensure valid UTF-8 is received.

Warning: When converting to or from UTF-8 (by using an encoding conversion such as kITKConvertFromISO_8859_1+kITKConvertToUTF8) and not using any transformation filters (such as kITKConvertToCR), it is possible that the length of the text will change before being sent. Therefore you cannot reliably use **BLOB size** to specify the number of bytes being sent. If the receiver needs to know exactly how many bytes to receive, then you must convert the blob to the “to” encoding.

For example:

```
// Instead of using kITKConvertFromUTF8+kITKConvertToISO_8859_1
$text:=Convert to text($blob;"utf-8")
CONVERT FROM TEXT($text;"iso-8859-1";$blob)

// Send a header to the receiver that indicates the length of the data being sent
sendHeader (BLOB size($blob))

$err:=ITK_SendBlob ($stream;$blob)
```

The *flushFlag* parameter is ignored, it has been ignored at least since ITK v3.5.

ITK_TCPSendBlob returns zero if successful, non-zero if an error occurred.

ITK_TCPRecvBlob
When using ITK_TCPRecvBlob and the received blob contains text, there are two encoding conversions: “from” conversion (the encoding in which the text is received over the network) and “to” conversion (the encoding of the text in the receiving blob). If no conversion is specified in the *filter* parameter (e.g. kITKConvertFromISO_8859_1), the blob data is considered binary data and is received as is. Otherwise, if you specify only one conversion (“from” or “to”), the other conversion defaults to the one you specify.

The encoding of *endString* is converted as follows:

- If a “to” encoding conversion is specified in the filter, that encoding is used.
- Else if a “from” encoding conversion is specified, that encoding is used.
- Else Mac Roman is used for backward compatibility.

For example, if you receive the data as UTF-8 and want to save it as ISO-8859-1, you would need to add kITKConvertFromUTF8+kITKConvertToISO_8859_1 to the filter.

Important: If you are receiving UTF-8 text in a blob, you should always add kITKConvertFromUTF8 to the filter, even if no encoding conversion is being done. This enables ITK to do special processing that helps to ensure valid UTF-8 is received.

If an error occurs, a non-zero value is returned.

ITK_TCPStatus, ITK_TCPStatus2A statuses:
New named constants have been added to represent all of the possible TCP statuses:

kStreamStatusInvalid = -1
kStreamStatusClosed = 0
kStreamStatusListen = 2
kStreamStatusSYNReceived = 4
kStreamStatusSYNSent = 6
kStreamStatusEstablished = 8
kStreamStatusFINWait1 = 10
kStreamStatusFINWait2 = 12
kStreamStatusCloseWait = 14
kStreamStatusClosing = 16
kStreamStatusLastACK = 18
kStreamStatusTimeWait = 20

ITK_TCPStatus
A new named constant has been added for use when you want to get the status of a stream:

kITKStreamStatus = 0

ITK_UDPSend / ITK_UPDRcv
Text is converted to UTF-8 when sending and from UTF-8 when receiving.

ITK_Addr2Name
New named constants for use with the *options* parameter:

kAddressResolutionAny = 0
kAddressResolutionName = 1
kAddressResolutionDotted = 2

ITK_Bin2Mac
For technical reasons, no attempt is made to preserve file creation/modification time.

ITK_Text2ISO / ITK_ISO2Text
ITK_Text2ISO and ITK_ISO2Text have been deprecated (they do nothing), because you can no longer store anything but Unicode in 4D text variables/fields. Any non-Unicode encoding must be stored in BLOBs. Here is sample code to translate from the deprecated routines:

```
// ITK_Text2ISO
CONVERT FROM TEXT($text; “ISO-8859-1”; $blob)

// ITK_ISO2Text
$text:=Convert to text($blob; “ISO-8859-1”)
```

To support other character sets, please refer to the 4D documentation for CONVERT FROM TEXT.

Text2URL / URL2Text
Text2URL/URL2Text always converts to/from UTF-8, so option 4 (do not apply ISO encoding) does nothing.

Text2HTML / HTML2Text
Text2HTML and HTML2Text will only work with Unicode characters with values 0x0000 to 0xFFFE, which encompasses most modern languages.

Text2B64 / B642Text
Text2B64 encodes the text as UTF-8 and then base64 encodes that. The reverse is done in B642Text. If invalid base64 characters are passed in B642Text, the result is an empty string.

Text2Quoted / Quoted2Text
Text2Quoted encodes the text as UTF-8 and then quoted printable encodes that. The reverse is done in Quoted2Text. So flag = 1 is now ignored.

ITK_EncryptText / ITK_DecryptText
Because encryption results in raw data which cannot safely be stored in a Unicode string, the encrypted text must be stored in a blob. The signature of these commands is now:

error := ITK_EncryptText (text; secretKey; algoID; blockMode; IV; blob {; utf8 })
error := ITK_DecryptText (blob; secretKey; algoID; blockMode; IV; text {; utf8 })

blob receives the encrypted text in ITK_EncryptText, the *text* parameter is left unchanged. The optional longint parameter *utf8*, if passed and non-zero, indicates that *text*, *secretKey* and *IV* should be converted to UTF-8 before encryption/decryption, and that decrypted text should be converted from UTF-8. Otherwise Mac Roman is used for backward compatibility.

Note: We recommend you pass *utf8* = 1 to ensure the full range of Unicode characters can be represented in the encrypted text.

ITK_EncryptBlob / ITK_DecryptBlob
A new*utf8* longint parameter has been added at the end. If not passed or zero, the text is converted to/from Mac Roman before being encrypted/decrypted. This ensures backward compatibility if you are storing encrypted blobs. If the *utf8* parameter is non-zero, the text is converted to/from utf8.

ITK_DigestAdd
A new longint parameter, *utf8*, has been added at the end. If passed and non-zero, the text is converted to UTF-8 before being added to the digest. Otherwise the text is converted to Mac Roman for backward compatibility.

Note: We recommend you pass *utf8* = 1 to ensure the full range of Unicode characters can be represented.

ITK_DigestCalc
Because Unicode text cannot contain arbitrary data, option zero for *digestFormat* (8-bit text) is no longer supported. If you do not pass *digestFormat* or pass zero, an empty string will be returned.

ITK_OpenFile
Unlike the other file-based commands, this command cannot use Posix-style paths on OS X.

Note: This command is completely superfluous, since it's exactly the same as **Open document(path; Read Mode)**. We recommend you use the native 4D command instead.

ITK_PictSize
Because 4D v11+ handles pictures differently than previous versions, *top* and *left* will always be zero. Also, if the picture is a PNG, *type* will be 4. Type 3 (Mac PICT containing JPEG) is deprecated. Note that 4D may store multiple representations of a picture in a single picture variable. This command always returns info on the first representation.

ITK_Pict2Blob / ITK_Blob2Pict
These are no longer a simple transformation of a picture into a blob, so a copy of the data is always made internally. Note that 4D may store multiple representations of a picture within a single 4D picture variable, and in the case of ITK_Pict2Blob, the first representation of the picture is returned.

ITK_Pict2GIF
This command is no longer supported. Instead you should install the method ITK_PictToGIF that is included with the plugin, and change plugin calls to ITK_Pict2GIF to method calls to ITK_PictToGIF. The signature of this method is:

```
gif := ITK_PictToGIF(pict {; flag {; width {; height {; transparentColor}}}})
```

Please note the following:

- Interlacing is no longer supported. You may pass the same value for *type* that was passed for *flag* in the old ITK command, but only the transparency bit is used.
- Instead of passing the x,y position of the transparent pixel, you now pass the RGB color (as a longint) of the color you want to be transparent.
- Transparency only works on 4D v14 R2 or later.
- Because the method must using scaling by a factor to resize the image, it is possible that the final image width or height may be off by one because of rounding errors.

ITK_PictSave
Previous versions of ITK on OS X used the Finder's type/creator info to identify the saved picture's type. Because file type/creator use is no longer recommended, ITK v4 adds the appropriate filename extension to the path if necessary. 4D may store multiple representations of a picture in a single picture variable; only the first representation is saved.

ITK_PictRead / ITK_PictSave
As with all other file-based commands, either an HFS or Posix-style path can be used on OS X. PNG is also supported as a picture type. The *options* parameter is still honored, but should be considered deprecated since PICT files are pretty useless these days.

ITK_BlobSearch / ITK_BlobReplace
By default, these commands convert the search/replace text to Mac Roman for backward compatibility. A new *utf8* longint parameter has been added at the end. If non-zero, the search text is converted to UTF-8, and thus assumes the blob text is UTF-8 as well.

ITK_RFC2Secs / ITK_Secs2RFC
Only dates between 1 Jan 1970 00:00:00 GMT and 19 Jan 2038 03:14:07 GMT are valid.

An extra longint parameter, *timeZoneNameFlag*, was added to ITK_Secs2RFC. If it is non-zero, the timezone name (if available) will be used instead of an offset. Note that if *timeZoneFlag* is 1 (local time), "GMT" is always used, even if *timeZoneNameFlag* is 0.