Internet Toolkit User Manual

Version 4.0

e-Node
30 rue de la République
33150 Cenon
France

www.e-node.net

Manual revision:5 Sept 2016

Contents

About Internet ToolKit

What is Internet ToolKit, and what can I do with it?

Internet ToolKit (ITK) is an inexpensive yet extremely powerful plug-in for 4D that allows you to transform any 4D database into an Internet server or client.

ITK can be used in all 4D products to provide complete TCP/IP connectivity for your application.

You can use ITK to implement FTP (File Transfer Protocol) servers, email (using SMTP and POP3), news servers, mailing list, and Web clients. You can even design your own custom TCP/IP protocols, to implement application-to-application communications like data synchronization thanks to its low-level approach.

Utility commands are provided to convert picture formats, encode files using MacBinary, BinHex or uuencode, and to convert text to HTML, with options for URL encoding and ISO character conversions, Base64, Quoted Printable, etc.

ITK adds 77 commands to 4D's language, enabling low and high level control of TCP/IP stacks, including UDP, DNS and SSL.

Technical Details

Compatibility Information

ITK version 4 is compatible with 4D v14, v15 and above, for both MacOS and Windows (including 32-bit and 64-bit servers). It requires MacOS 10.7.5 or higher and Windows 7 or better.

> You do not have to update all your ITK code. The commands are still here and will work with ITK version 4 with little or no change in your code, except for all Unicode related commands (since this version is full Unicode), and a few deprecated or obsolete commands that have been removed. See the Upgrading from Previous versions section.

Use ITK version 3.5 with 4D v11, v12 and v13, and ITK v2.6 with pre-v11 versions such as 4D 2004. License keys for any of these two versions will be provided upon request when purchasing any ITK license.

Technical Support

Technical support for ITK is provided electronically via the online web forums.

> Note: many examples with their source code are available in the ITK Demonstration and Test databases.

2 Installation

Installing the plugin

ITK is provided as a bundle for both Windows and MacOS: there is just one version for both platforms. To install it, simply copy the file **ITK.bundle** into your Plugins folder.

Plugins folders can be located in one of two locations:

- In the 4D application folder (4D or 4D Server). When plugins are installed in this location, they will be available to every database that is opened with that application.
- Next to the database structure file for your project: in this case, the plugin will only be available to that database. On MacOS, this means that the Plugins folder must be placed within the database package or folder. To open a package, ctrl-click on the package and choose **Show Package Contents** from the contextual menu.

Using ITK in Demo mode

You can use ITK in Demo mode for 20 minutes, after which time it will cease to work. When this becomes annoying, it's time to buy a license, which you can do on our website.

Licenses are either linked to the 4D product number, the workstation or the company name as described below.

Licensing

Like all e-Node plug-ins, ITK offers several license types. There are no such things as MacOS vs Windows or Development vs Deployment.

For current pricing, please see the ordering page on our website.

Definitions

- **Regular licenses** are used for applications that are opened with 4D Standalone or 4D SQL Desktop, or with 4D Server, either in interpreted or compiled mode (doesn't make a difference regarding plugin licensing).

  These can be either single user or server databases and they are linked to the 4D or 4D Server license: you need to provide the number returned by the "Copy" or "eMail" buttons from the plugin demonstration mode alert (this number is actually the 4D command **GET SERIAL INFORMATION** first parameter). This number is a negative long integer such as -1234567.

- **Merged licenses** are used for double-clickable applications built with 4D Volume Desktop (single user) or with 4D Server by means of the 4D Compiler module.

  These licenses are linked to the machine ID (single user workstation or server): you need to provide the number returned by the "Copy" or "eMail" buttons from the plugin demonstration mode alert (this number is calculated from the single user or server machine UUID). On 4D Server any remote client will return the server number. This number is a positive long integer such as 1234567.

In both cases the demonstration mode dialog will display the proper number according to the current setup (regular or merged) and the "Copy" and "eMail" buttons will use it as well.

Free updates

- **Regular licenses**. A new license will be supplied for free at any time (maximum once a year) if you change your 4D version or get a new 4D registration key for the same version, provided that your previous license match the current public version at exchange time. This rule applies whether you are already using the new version or not: just specify that you also want a key for the older version as well as the current one when you order an upgrade.

- **Merged licenses**. These licenses are independent from the 4D versions and product numbers. They will remain functional if you upgrade e.g. from 4D v14 to 4D v15 on the same machine (single user workstation or server).

  You'll only need to update a merged license if your machine or motherboard is replaced (a new license will be supplied for free in this case, provided that your previous license match the current public version at the exchange time), or if you install a paid upgrade of the plugin.

Note: if you are using several concurrent versions of 4D you will need one plugin license for each version.

License types

- **Single-user.** This license allows development (interpreted mode) or deployment (interpreted or compiled mode, including merged) of applications that are opened with 4D Standalone or 4D SQL Desktop or built with 4D Volume Desktop.
- **Server.** These licenses allow development (interpreted mode) or deployment (interpreted or compiled mode, including merged servers / remotes) on 4D Server with up to 10 users (“small server”), 11 to 20 users (“medium server”) or more (“large server”).
- **Unlimited Single User.** This license allows development (interpreted mode) or deployment (interpreted or compiled mode, including merged) on any number of 4D Standalone (or single user merged applications built with 4D Volume Desktop) that run your 4D application(s).

  It is a yearly license, which expires after the date when it is to be renewed. Expiration only affects interpreted mode. **Compiled applications using an obsolete license will never expire**.

  A single license key will unlock all setups on all compatible 4D versions and all versions of the plugin. The license key is linked to the developer/company name.

  This license allows deployment (selling new application licenses, updates or subscriptions) while the license is valid. **No new deployment may occur after expiry without a specific license** (merged or regular). End-users running deployments sold during the license validity period remain authorized without time limit, provided that they are no longer charged for the application using the plug-in (including maintenance or upgrades).
- **OEM.** This license allows development (interpreted mode) or deployment (interpreted or compiled mode, including merged) on any number of 4D Servers (any number of users), 4D Standalone or single user/remote merged instances that run your 4D application(s).

  It is a yearly license, under the exact same terms as the Unlimited Single User license described above, except that it also covers server deployments.
- **Unlimited OEM.** This license is a global OEM license, covering any combination of the plug-ins published by e-Node, including AreaList Pro, SuperReport Pro, PrintList Pro, CalendarSet and Internet ToolKit in all configurations.
- **Partner license.** This license matches 4D’s annual Partner subscription and covers all the plug-ins published by e-Node, including AreaList Pro, SuperReport Pro, PrintList Pro, CalendarSet and Internet ToolKit.

  For each product, a single registration key allows development (interpreted mode) or deployment (interpreted or compiled mode, except merged) on all 4D Standalones and 4D Servers (2 users) regardless of 4D product numbers, OS and versions. No merged applications.

  This is a yearly license, expiring on February 1st (same date as 4D Partner licenses). Expiration only affects interpreted mode. **Compiled applications using an obsolete license will never expire**.

Note: you don’t have to be a 4D Partner subscriber to subscribe to the e-Node Partner license.

Registering your ITK License

Once you have purchased your license, you will receive a registration key. This code must be registered each time the database is started. There are four ways to register your license:

- using the Demo mode dialog “Register” button,
- though a text file,
- in your 4D code with a command,
- through the online automated registration system.

Yearly licenses such as Unlimited single user, OEM and Partner do not require any serial information or online registration. The only way to register these licenses is through the ITK_Register command.

Quick and easy way

ITK V4

1. Put the following lines of code into your **On Startup** database method, with the license number that you received and your email address:

```
C_LONGINT ($result)
$result:=ITK_Register ("yourLicenseKey";0;"youremail@something.xxx")  // 0 if successful
```

2. Make sure that the machine where the plugin will be used is connected to the Internet (single user workstation or in server mode the first remote client that will connect to the server).
3. Install your application.
4. The plugin will silently (no dialog) register itself.
5. You will receive an email with the final key issued and the IP address of the user site.

If the site has no Internet connection or if you want to use the plugin license system to help protect your own software copy, you can manage the final key registration yourself using one of the following methods.

The Demo mode dialog

Single user and server licenses require that you first send us the relevant information (serial or machine ID, see Definitions), unless you are using the Online registration system.

This is performed from the demonstration mode dialog.

The Demo mode dialog is displayed upon the first call to ITK (through a command).

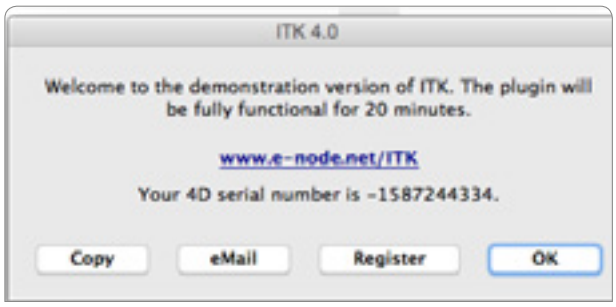
To trigger this display and enable your users to register without actually calling a command or setting up an area, pass an **empty** string to ***ITK_Register*** and the dialog will show:

```
C_LONGINT ($result)
$result:=ITK_Register ("")  // display the dialog
```

Note: calling ITK_Register with any key (valid or invalid) will not display the dialog.

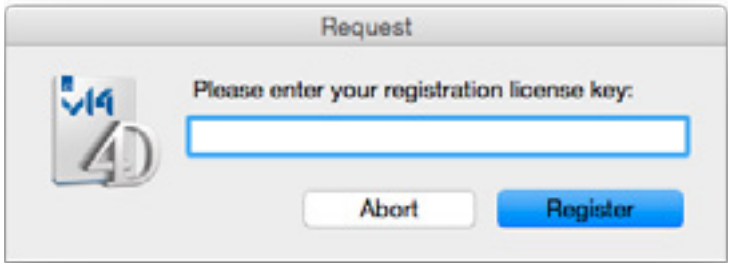
Retrieving the serial/machine information

The Demo mode dialog includes all relevant information (serial or machine ID, see Definitions) to obtain your license, as well as a “Copy” button to put this information into your clipboard or a text file, an “eMail” button to email the information to e-Node’s registration system and a “Register” button to enter your license key once received:

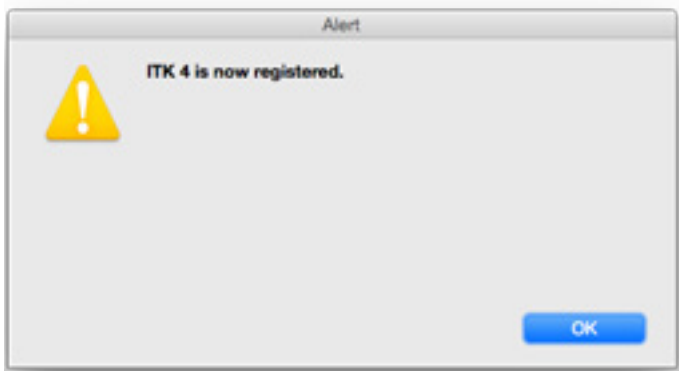


Using the “Register” button

Clicking on this button will display a standard 4D request to enter your registration key:



Paste or drag and drop your registration key and, if correct, the plug-in will be registered for all future uses on this workstation:



Note: if 4D does not activate the **Edit > Paste** menu item click **Abort** and **Register** again, or try drag and drop.

Registering Server licenses

Similarly, server licenses can be registered from the demonstration mode dialog without having to modify your code and use ITK_Register (which of course you can do with any license type). In this case, the 4D Licenses folder, serial information or machine ID used will only be the 4D Server information, not the client workstation's.

Server licenses can be registered on any client workstation (remote mode), or on 4D Server itself.

Registering in Remote mode

The server and all workstations can be registered from any single client workstation connected to the server. As in Single user mode, the Demo mode dialog will be displayed on a client workstation when one of the following conditions are met:

- Calling an ITK command other than ***ITK_Register*** with a non-empty parameter
- Calling ***ITK_Register*** with an empty string

Use the **Copy**, **eMail** and **Register** buttons just as above and your server will be registered for all workstations.

Note: any other workstations previously connected (before registration occurred) will need to re-connect to the server to be functional.

Registering on 4D Server

To directly register the server and all workstations from the server machine itself, you need to display the Demo mode dialog on the server.

Call ***ITK_Register*** with an empty string in the **On Server Startup** base method:

```
C_LONGINT ($result)
$result:=ITK_Register ("")  // display the dialog
```

Use the **Copy**, **eMail** and **Register** buttons just as above and your server will be registered for all workstations.

Note: the dialog will automatically be dismissed on the server after one minute in order not to block client connections (the server is only available to client workstations once the On Server Startup method has completed).

Merged licenses notes

Both methods can be either used with regular or merged servers and client workstations.

- Regular licenses are linked to the 4D Server serial information
- Merged licenses are linked to the 4D Server machine ID

Note: merged licenses will keep working if your 4D Server serial information is modified (upgrading or 4D Partner yearly updates), or if any client workstation hardware is changed.

It will only need to be updated if the 4D Server hardware is changed, or if the plugin itself requires a new key (paid upgrades upon major version changes).

You may want to register your merged server without having to turn off the database to modify the code. We have created a utility database to manage this - it's called Get Serial Info and you can download the appropriate version for your 4D version from the e-Node server.

This is possible using any 4D setup on the server machine (such as a standard developer single user 4D). Keeping your production server alive, open the Get Serial Info database with 4D on the same server machine. Ignore the demonstration mode dialog (if your single user 4D is not registered for the plugin) and wait for the next Alert:

A text file is also saved with the same information.

The last line "Machine ID" is the number that you need to send in order to receive your merged server registration key.

You can also check the machine ID in standalone mode (or on any remote client with the built-client application or in interpreted mode as long as it is running on the same server machine) with AreaList Pro using the following call:

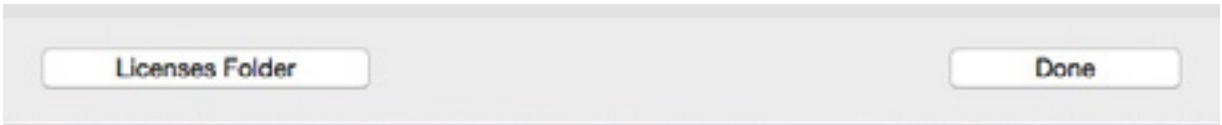
```
C_LONGINT($machineID)
$machineID:= AL_GetAreaLongProperty (0;"mach")
```

Note: you don't need an AreaList Pro license to do this.

Using a text file

Alternately, you can place a plain text file into your 4D Licenses folder.

To open this folder from 4D use the 4D Menu **Help > Update licenses,** then click the **Licenses Folder** button:



The text file **must** be called "ITK4.license4Dplugin" and be a plain text type file.

Just paste all your licenses for ITK v4.x, one per line, e.g.:

```
MyLicense1
MyLicense2
MyLicense3
```

Any license type can be included into this document, except unlimited single user, OEM and Partner licenses.

Note: the Demo mode dialog **Register** button actually does this: create the text file and include the license key, or add the license key to the existing document if any.

Using ITK_Register

1. Open the **On Startup** database method

2. Call the ITK_Register function with your registration key - for example:

```
$result:=ITK_Register ("YourRegistrationKey")  // result = 0 means registration was successful
```

If you have several licenses for different 4D setups you can call ***ITK_Register*** multiple times in a row without further testing. See the Example with multiple calls.

Combining methods

When such a file exists in the Licenses folder ITK will check for valid licenses from this document as a first action before anything else (including checking any ***ITK_Register*** command).

If a valid license is included into the “ITK4.license4Dplugin” document any calls to ***ITK_Register*** will return zero (for “OK”).

Therefore you can mix modes and use the text file (or **Register** button) as well as the command.

Unlimited single user, OEM, temporary and Partner licenses can only be entered through the ***ITK_Register*** command.

ITK V4

Online registration

As of version 4.0, ITK provides an automated solution to register itself using an Internet connection.

This feature can be helpful whenever you don’t want to bother your end user with plugin registration, or want to save the time to collect the serial / machine ID, or any other reason when you want the process to be entirely and automatically managed from the client site.

It can also be used for your own development tools, removing the need to modify your 4D code to include or update registration licenses.

Note however that the site must have an open outgoing HTTP Internet connection available.

“Master” keys

The basic principle is that we deliver a non-assigned license key, called “master key”, which you use in your call to ITK_Register in your **On Startup** database method. This key will be used to generate valid keys for the plugin and environment, called “final keys”.

One single master key can generate as many final keys as you need, in case you order several licenses of the same kind (regular or merged, single user licenses or server licenses of the same size).

A master key looks like a final key, except that the second part is the plugin code name (same as the license file name) instead of the serial / machine ID, e.g. “123456-ITK4-xyz”.

Passing a master key as the first parameter to ***ITK_Register*** when the plugin has not been previously registered by any of the methods above will result in a connection attempt to e-Node’s license server as described below.

Process

If the plugin has not been previously registered (through online registration, text file, register button or ITK_Register with a final key), and if ***ITK_Register*** receives a master key in its first parameter, it will recognize it a such, then:

1. Connect to e-Node's license server.

2. Ask the server if the master key has not been assigned yet (or if the master key is designed to generate several final keys, if there is any unassigned key up to that number).

3. Send the serial information (regular licenses) or the machine ID (merged licenses) to the license server.

4. If an error is detected (such as master key not matching the current setup) return an error to ***ITK_Register***.

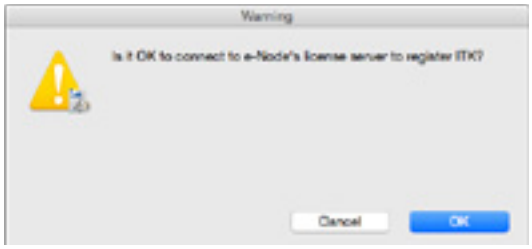
5. If the master key is valid, receive its final key from the license server then register itself (writing into the license file).

Note: if a final key has already been issued for this serial / machine ID using this master key, it is simply resent.

User interface

In addition, ITK_Register second parameter allows optional settings regarding the user interface in the online registration process.

Display a confirmation dialog before step 1



Display an alert at step 4

Display an alert at step 5

e-Mail notification

The third parameter to ITK_Register (optional) is the developer email to whom the information will be sent (if this parameter is used and non empty, of course).

The emailed information includes both the final key issued and the IP address from where it was requested (and to where it was sent for registration).

- When a key is issued:

  Title: ITK4 license

  Body:

  License 123456-123456789-abcdefgh

  issued to 12.34.56.78

- When a key is resent:

  Title: ITK4 license

  Body:

  License 123456-123456789-abcdefgh

  resent to 12.34.56.78

The default mode (master key being passed as the only parameter) is silent: no confirmation, no alert, no email.

Getting Started with ITK

Working with ITK Commands and Functions

Each command you write must adhere to a specific syntax in order for it to be correctly understood by ITK. Some commands return a **result**: these are functions You can use the commands and functions to configure every operation performed by ITK, and to get various informations.

Each ITK command has a syntax, or rules, that describe how to use the command in your 4D database. For each command, the name of the command is followed by the command's parameters, and result in case of functions.

An ITK command syntax looks like this:

ITK_Pict2Blob

(picture:P; option;L) ➞ result:O

The parameters are enclosed in parenthesis, and separated by semicolons.

Each parameter is followed by a colon and a letter indicating the type of data required for that parameter:

:I - integer

:L - longint

:O - blob

:P - picture

:R - real

:T - text

:Y - array

:Z - pointer

Each is preceded by one of three arrow signs, which indicate whether it is a value that you pass to the command or one that the command returns to you, or a value that is passed, then modified and returned by the command in the same parameter:

➞	parameter	A value that you pass to the command
⬅	parameter	A value that is returned by the command
⇆	parameter	A value that is passed to, then returned by the command

The commands and functions are grouped into themes: ITK Blob Commands, ITK Conversion Commands, ITK Encryption Commands, ITK SSL Commands, ITK TCP/IP Commands, ITK UDP Commands and ITK Utility Commands.

Note: when calling a plugin command, all omitted parameters are initialized to the NULL of the respective types (0, 0.0, "", !00:00:00!, …).

Upgrading from Previous versions of ITK

ITK version 4 is compatible with 4D version 14 and above.

To upgrade to ITK version 4, simply install it as described in the Installation section of this manual, replacing your older version.

What's New

Online registration

As of version 4.0, ITK provides an automated solution to register itself using an Internet connection.

This feature can be helpful whenever you don't want to bother your end user with plugin registration, or want to save the time to collect the serial/machine ID, or any other reason when you want the process to be entirely and automatically managed from the client site. See Online registration in the Installation chapter.

64-bit Server

ITK v4 supports 64-bit server on Windows and MacOS.

Unicode

ITK v4 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 2 GB, 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 682 MB.

What's Changed

Command parameters and behavior

Many commands have been enhanced in ITK V4. Look for the ITK V4 pictogram in the command chapters.

Filters

The "old" set of filters is as follows:

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

Some of these filters are character encoding conversions, and some are text transformations.

In ITK v4, 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.

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

If a “ToISO/UTF8” conversion is passed to a receive command, or a “FromISO/UTF8” conversion is passed to a send command, or ISO and UTF8 conversions are both passed, kITKInvalidConversion (-7) is returned.

Paths

All commands that take a file path (except ITK_OpenFile) can now take a Posix-style path on MacOS X.

For example, instead of “System:Users:homer:Documents:marge.jpg”, you can use “/Users/homer/Documents/marge.jpg”.

If a MacOS X path contains “:”, it is assumed to be an HFS path, otherwise it is assumed to be a Posix 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.

Encryption

AES (also known as Rijndael) 256 encryption is now supported. See Encryption Commands.

Obsolete Commands

The following commands can be found under the _ITK Obsolete Commands theme in the design mode explorer.

Many of them have been reported as completely unused by all users in the survey that we ran prior to rewriting ITK.

Note: these commands are now obsolete and do nothing.

_ITK Obsolete Commands theme

Old Command	Reason
ITK_Blobgunzip	Reported as unused
ITK_Blobgzip	Reported as unused
ITK_BlobRead	Reported as unused
ITK_BlobSave	Reported as unused
ITK_CountIPC	Reported as unused
ITK_GetIndIPC	Reported as unused
ITK_GetPriority	Reported as unused
ITK_ICCountKeys	Internet Config commands are no longer supported
ITK_ICGetIndKey	Internet Config commands are no longer supported
ITK_ICGetPref	Internet Config commands are no longer supported
ITK_ICGetRaw	Internet Config commands are no longer supported
ITK_ICMPEcho	Internet Config commands are no longer supported
ITK_ICSetPref	Internet Config commands are no longer supported
ITK_ICSetRaw	Internet Config commands are no longer supported
ITK_ISO2Text	Obsolete, because you can no longer store anything but Unicode in 4D text variables/fields. Any non-Unicode encoding must be stored in BLOBs using **CONVERT FROM TEXT** and **Convert to text** (see the note below)
ITK_Mac2uu	Reported as unused
ITK_Pict2GIF	Obsolete, use the provided 4D method (see below)
ITK_PPPClose	PPP commands are no longer supported
ITK_PPPOpen	PPP commands are no longer supported
ITK_PPPStatus	PPP commands are no longer supported
ITK_SetPriority	Reported as unused
ITK_Text2ISO	Obsolete, because you can no longer store anything but Unicode in 4D text variables/fields. Any non-Unicode encoding must be stored in BLOBs using **CONVERT FROM TEXT** and **Convert to text** (see the note below)
ITK_TimerLap	Reported as unused
ITK_TimerStart	Reported as unused
ITK_TimerStop	Reported as unused
ITK_uu2Mac	Reported as unused

ITK_Pict2GIF replacement 4D project method

ITK_Pict2GIF is no longer supported. Instead you should install the method ***ITK_PictToGIF*** that is included in the Test database, 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.

The code to this 4D project method is as follows:

```
//ITK_PictToGIF($pict {; $flag {; $width {; $height {; $transparentColor}}}}) -> gifPict
C_PICTURE($1;$pict;$0)
C_LONGINT($2;$flag;$3;$width;$4;$height;$5;$color)
$pict:=$1
If (Count parameters>1)
   $flag:=$2
Else
   $flag:=0
End if
If (Count parameters>2)
   $width:=$3
Else
   $width:=0
End if
If (Count parameters>3)
   $height:=$4
Else
   $height:=0
End if
If (Count parameters>4)
   $color:=$5
Else
   $color:=0x00FFFFFF
End if
C_LONGINT($currentWidth;$currentHeight)
PICTURE PROPERTIES($pict;$currentWidth;$currentHeight)
C_REAL($widthFactor;$heightFactor)
```

```
If ($width=0) | ($currentWidth=0)
   $widthFactor:=1
Else
   $widthFactor:=$width/$currentWidth
End if
If ($height=0) | ($currentHeight=0)
   $heightFactor:=1
Else
   $heightFactor:=$height/$currentHeight
End if
CONVERT PICTURE($pict;"gif")
If (($flag & 2)#0)
   C_TEXT($version)
   $version:=Application version
   If (Num(Substring($version;1;2))>=14) & ($version[[3]]>="2")
      // 102 = Transparency
      TRANSFORM PICTURE($pict;102;0x00FFFFFF)
   End if
End if
If ($widthFactor#1) | ($heightFactor#1)
   TRANSFORM PICTURE($pict;Scale;$widthFactor;$heightFactor)
End if
$0:=$pict
```

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 commands:

```
// 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**.

ITK Blob commands

ITK_Blob2Pict

(blob:0; option;L) ➞ result:P

Parameter	Type	Description
➞ blob	blob	Blob to convert.
➞ option	longint	Option to indicate if a copy of the **blob** is requested.
⬅ result	picture	Converted picture.

ITK_Blob2Pict converts a **blob** value (variable or field) into a picture.

option — longint. This value indicates whether (0) or not (1) a copy of the **blob** is requested.

Mode	Value
A copy of the **blob** is returned as a picture	0
A copy of the **blob** is returned as a picture, and the **blob** is emptied	1

ITK V4

Note: this command is no longer a simple transformation from a blob handle to a picture handle, so a copy of the data is always made internally.

Examples

```
$myPict:= ITK_Blob2Pict($myBlob)  //a copy of the blob is returned as a picture, $myBlob is unchanged
$myPict:= ITK_Blob2Pict($myBlob;0)  //same as above
$myPict:= ITK_Blob2Pict($myBlob;1)  //a copy of the blob is returned as a picture, $myBlob is emptied
```

ITK_Blob2Record

(blob:0; tableNumber:L) ➞ error:L

Parameter	Type	Description
➞ blob	blob	Blob to convert.
➞ tableNumber	longint	Table number.
⬅ result	longint	4D error code

ITK_Blob2Record converts a blob containing a record obtained by ITK_Record2Blob back into a 4D record (in the current selected record).

tableNumber — longint. This is the value returned by the 4D **Table** command.

Example

```
CREATE RECORD([File1])
$err:=ITK_Blob2Record($blob;Table(->[File1]))
SAVE RECORD([File1])
```

ITK_BlobReplace

(blob:0; offset:L; oldText:T; newText:T; nbOcc:L; utf8:L) ➞ position:L

Parameter	Type	Description
➞ blob	blob	Blob to convert.
➞ offset	longint	Start search position in the **blob**.
➞ oldText	text	Text to search in the **blob**.
➞ newText	text	Text that will replace **oldText**.
⇆ nbOcc	longint	Number of occurences to search/replace.
➞ utf8	longint	Use UTF-8.
⬅ position	longint	Last position found (starting at **offset**) or -1 if **oldText** was not found in **blob**.

ITK V4

ITK_BlobReplace searches a text in a **blob** from a given position (**offset**), and replaces it with the specified new text.

Note: text matching is byte based. This means that "E", "e" and "é" are considered different.

offset — longint.

Position	Value
Start of the blob (first byte in **blob**)	0
Second byte in **blob**	1
etc.	

nbOcc — longint. Can be used to repeat this search/replace several times.

Mode	Value
All occurences	-1
First occurence	0 or 1
etc.	

Note: the **nbOcc** parameter will also return the number of occurences actually replaced.

ITK V4

utf8 — longint. By default, ***ITK_BlobReplace*** converts the search/replace text to Mac Roman for backward compatibility. This new parameter was added in version 4: if non-zero, the search text is converted to UTF-8, and thus assumes the blob text is UTF-8 as well.

Example

```
TEXT TO BLOB("Test text";$b;3)
$p := ITK_BlobReplace($b;0;"te";"Te")  // $p returns 5, blob now contains "Test Text"
```

ITK_BlobSearch

(blob:0; offset:L; text:T; nbOcc:L; utf8:L) ➡ position:L

Parameter	Type	Description
➡ blob	blob	Blob to convert.
➡ offset	longint	Start search position in the **blob**.
➡ text	text	Text to search in the **blob**.
⇆ nbOcc	longint	Number of occurences to search.
➡ utf8	longint	Use UTF-8.
⬅ position	longint	Last position found (starting at **offset**) or -1 if **text** was not found in **blob**.

ITK_BlobSearch searches a **text** in a **blob** from a given position (**offset**), and returns the position of the Nth occurence of that text (N being the **nbOcc** parameter).

Can also be used to count the number of occurences of a given text into a blob (see example).

Note: text matching is byte based. This means that "E" and "e" and "é" are different.

offset — longint.

Position	Value
Start of the blob (first byte in **blob**)	0
Second byte in **blob**	1
etc.	

nbOcc — longint. Can be used to repeat this search several times.

Mode	Value
All occurences	-1
First occurence	0 or 1
etc.	

Note: the **nbOcc** parameter will also return the number of occurences found.

ITK V4

utf8 — longint. By default, ***ITK_BlobReplace*** converts the search text to Mac Roman for backward compatibility. This new parameter was added in version 4: if non-zero, the search text is converted to UTF-8, and thus assumes the blob text is UTF-8 as well.

Examples

```
TEXT TO BLOB("Test text";$b;3)
$p := ITK_BlobSearch($b;0;"text")  // returns 5

 // count occurences
$nb := BLOB size($b)  // max number of occurences to count (up to blob size)
$p := ITK_BlobSearch($b;0;"e";$nb)  // $p returns 6, $nb returns 2 (two "e"s found" in blob)
```

ITK_Pict2Blob

(picture:P; option;L) ➞ result:O

Parameter	Type	Description
➞ picture	picture	Picture to convert.
➞ option	longint	Option to indicate if a copy of the **picture** is requested.
⬅ result	blob	Typecasted blob.

ITK_Pict2Blob converts a **picture** value (variable or field) into a blob.

ITK V4

Note: 4D may store multiple representations of a picture within a single 4D picture variable: in this case the first representation of the picture is converted.

option — longint. This value indicates whether (0) or not (1) a copy of the **picture** is requested.

Mode	Value
A copy of the **picture** is returned as a blob	0
A copy of the **picture** is returned as a blob, and the **picture** is emptied	1

ITK V4

Note: this command is no longer a simple transformation from a picture handle to a blob handle, so a copy of the data is always made internally.

Examples

```
$myBlob:= ITK_Pict2Blob($myPict)  //a copy of the picture is returned as a blob, $myPict is unchanged
$myBlob:= ITK_Pict2Blob($myPict;0)  //same as above
$myBlob:= ITK_Pict2Blob($myPict;1)  //a copy of the picture is returned as a blob, $myPict is emptied
```

ITK_Record2Blob

(tableNumber:L; blob:0) ➞ error:L

Parameter	Type	Description
➞ tableNumber	longint	Table number.
⬅ blob	blob	Blob containing the converted record.
⬅ result	longint	4D error code

ITK_Record2Blob converts a the current record of the specified table into a **blob**, which can eventually be saved back into a 4D record using ITK_Blob2Record.

tableNumber — longint. This is the value returned by the 4D **Table** command.

Example

```
GOTO SELECTED RECORD([File1];1)
$err:=ITK_Record2Blob($blob;Table(->[File1])
```

ITK Conversion Commands

ITK_B642Text

(base64text:T) ➞ result:T

Parameter	Type	Description
➞ base64text	text	Base64 text to decode.
➟ result	text	Decoded text.

ITK_B642Text decodes Base64 text. The Base64 conversion is a 8-bit to 6-bit conversion (4 chars decoded into 3 chars). The resulting text will be 25 % smaller than the original.

Note: ITK_Text2B64 encodes the text as UTF-8 and then base64 encodes that. The reverse is done in ***ITK_B642Text***. If invalid base64 characters are passed in ***ITK_B642Text***, the result is an empty string.

Example

```
$pass := ITK_B642Text($B64pass)
```

ITK_Bin2Mac

(binPath:T; filePath:T) ➞ result:L

Parameter	Type	Description
➞ binPath	text	Pathname of the MacBinary file to decode.
➞ filePath	text	Pathname of the resulting file.
⬅ result	longint	OS error code.

ITK_Bin2Mac decodes a MacBinary (8-bit) file into an original Mac or Windows file.

ITK V4

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

binPath, filePath — text. ITK_Mac2Bin and ***ITK_Bin2Mac*** can take Posix-style paths (using "/" as the directory separator) on MacOS X. If the path contains ":" and does not contain "/", it is assumed to be an HFS path.

Note: files saved with ITK_Mac2Bin are forward compatible (v3.5 can be converted with ***ITK_Bin2Mac*** in v4) but not backward compatible.

Examples

```
$err := ITK_Bin2Mac("MyHD:MyFolder:Myfile.bin";"MyHD:MyFolder:MyFile")  // MacOS
$err := ITK_Bin2Mac ("C:\\MyHD\\MyFolder\\Myfile.bin";"C:\\MyHD\\MyFolder\\MyFile")  // Windows
```

ITK_Date2Secs

(date:D; time:H; timeZoneFlag:L) ➞ gmtValue:L

Parameter	Type	Description
➞ date	date	Date to convert.
➞ time	time	Time to convert.
➞ timeZoneFlag	longint	Indicates in which time zone the date and time are provided.
⬅ gmtValue	longint	GMT date/time value.

ITK_Date2Secs converts a date and time (4D formats) into an GMT date/time value.

You can specify if the provided date and time are in local time zone or GMT.

ITK V4

Note: in ITK version 4, the only valid dates are between January 1, 00:00:00, 1970, UTC and January 19, 3:14:07, 2038, UTC, and the values returned by ITK_RFC2Secs and ***ITK_Date2Secs*** no longer match the values from the old version.

timeZoneFlag — longint:

Zone	Value
GMT time zone	0
Local time zone	1

Note: if you pass a local date and time (like **Current Date** and **Current Time**), be sure to specify local time zone in the **timeZoneFlag**, otherwise the resulting **gmtValue** will not be correct according to your local time zone and GMT time zone.

Examples

```
$secs := ITK_Date2Secs (Current date; Current time; 1)  //local time (in local time zone)
$secs := ITK_Date2Secs (!12/25/2015!;?00:00:00?;0)  //Christmas date, GMT time
```

ITK_gzip2Mac

(gzipPath:T; filePath:T) ➞ error:L

Parameter	Type	Description
➞ gzipPath	text	Pathname of the zip compressed file.
➞ filePath	text	Pathname of the decompressed file.
⬅ error	longint	Error code.

ITK_gzip2Mac decompresses a gzip compressed file (normally ending with ".gz") into an original Mac or Windows file.

gzip is a standard compressed file format defined in RFC #1952 which uses the deflate compression algorithm (based on LZ77 and Huffman coding, see RFC #1951) and contains a CRC to detect data corruption (see RFC#1950).

gzip compression can also be used in HTTP to send compressed data (see RFC #1945 sections 3.5, 10.3 and D.2.3 for explanation on the use of gzip in HTTP/1.0 and RFC #2616 sections 3.5, 14.3 and 14.11 for explanation on the use of gzip in HTTP/1.1).

ITK V4

gzipPath, **filePath** — text. Can take Posix-style paths (using "/" as the directory separator) on MacOS X. If the path contains ":" and does not contain "/", it is assumed to be an HFS path.

error — longint:

Error	Value
No error	0
Error while opening the original file	-1
Error while creating / opening the compressed file	-2
Error while writing the decompressed file	-3
Error while closing the decompressed file	-4
Error while closing the compressed file	-5
Error while reading the compressed file	-6

Examples

```
$err := ITK_gzip2Mac("MyHD:MyFolder:Myfile.gz";"MyHD:MyFolder:MyFile")  // MacOS file
$err := ITK_gzip2Mac("MyHD:MyFolder:Myfile.gz";"MyHD:MyFolder:MyFile.temp")
$err := ITK_gzip2Mac("C:\\MyHD\\MyFolder\\Myfile.gz";"C:\\MyHD\\MyFolder\\MyFile")  // Windows
```

ITK_Hqx2Mac

(binhexPath:T; filePath:T) ➞ error:L

Parameter	Type	Description
➞ binhexPath	text	Pathname of the BinHex file.
➞ filePath	text	Pathname of the resulting file.
⬅ error	longint	OS error code.

ITK_Hqx2Mac decodes a BinHex 4.0 file into an original Mac or Windows file.

ITK V4

binhexPath, **filePath** — text. Can take Posix-style paths (using "/" as the directory separator) on MacOS X. If the path contains ":" and does not contain "/", it is assumed to be an HFS path.

Examples

```
$err := ITK_Hqx2Mac("MyHD:MyFolder:Myfile.hqx";"MyHD:MyFolder:MyFile")  // MacOS
$err := ITK_Hqx2Mac ("C:\\MyHD\\MyFolder\\Myfile.hqx";"C:\\MyHD\\MyFolder\\MyFile")  // Windows
```

ITK_HTML2Text

(HTMLText:T; option:L) ➞ result:L

Parameter	Type	Description
➞ HTMLText	text	HTML text to convert.
➞ option	longint	Conversion option.
⬅ result	text	Converted text.

ITK_HTML2Text converts HTML text to plain text. By default, all HTML special characters like "<", ">", "&" are converted unless **option** is set to 1. All 8-bit characters are always converted (e.g. é ➞ é).

ITK V4

Note: ITK_Text2HTML and ***ITK_HTML2Text*** will only work with Unicode characters with values 0x0000 to 0xFFFE, which encompasses most modern languages.

option — longint:

Error	Value
Full HTML conversion	0
Reduced HTML conversion ("<", ">" , "&" and """ are not converted	1
Extract text (remove HTML tags)	2

Examples

```
$myHTMLText := "&lt;B&gt;&eacute;&lt;/B&gt;"
$myText := ITK_HTML2Text($myText)  // returns "<B>é</B>"
$myText := ITK_HTML2Text($myText; 1)  // returns "&lt;B&gt;é&lt;/B&gt;"
$myText := ITK_HTML2Text($myText; 2)  // returns "é" (tags removed)
```

ITK_Mac2Bin

(filePath:T; binPath:T) ➞ result:L

Parameter	Type	Description
➞ filePath	text	Pathname of the Mac or Windows file to encode.
➞ binPath	text	Pathname of the resulting MacBinary file.
⬅ result	longint	OS error code.

ITK_Mac2Bin encodes a Mac or Windows file into MacBinary (8-bit) format.

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

ITK V4

filePath, binPath — text. ***ITK_Mac2Bin*** and ITK_Bin2Mac can take Posix-style paths (using "/" as the directory separator) on MacOS X. If the path contains ":" and does not contain "/", it is assumed to be an HFS path.

Note: files saved with ***ITK_Mac2Bin*** are forward compatible (v3.5 can be converted with ***ITK_Bin2Mac*** in v4) but not backward compatible.

Examples

```
$err := ITK_Mac2Bin("MyHD:MyFolder:Myfile.bin";"MyHD:MyFolder:MyFile")  // MacOS
$err := ITK_Mac2Bin ("C:\\MyHD\\MyFolder\\Myfile.bin";"C:\\MyHD\\MyFolder\\MyFile")  // Windows
```

ITK_Mac2gzip

(filePath:T; gzipPath:T; level:L) ➞ error:L

Parameter	Type	Description
➞ filePath	text	Pathname of the original file.
➞ gzipPath	text	Pathname of the gzip compressed file.
➞ level	longint	Compression level.
⬅ error	longint	Error code.

ITK_Mac2gzip compresses a Mac or Windows file into gzip format (normally ending with ".gz").

gzip is a standard compressed file format defined in RFC #1952 which uses the deflate compression algorithm (based on LZ77 and Huffman coding, see RFC #1951) and contains a CRC to detect data corruption (see RFC#1950).

gzip compression can also be used in HTTP to send compressed data (see RFC #1945 sections 3.5, 10.3 and D.2.3 for explanation on the use of gzip in HTTP/1.0 and RFC #2616 sections 3.5, 14.3 and 14.11 for explanation on the use of gzip in HTTP/1.1).

ITK V4

filePath, **gzipPath** — text. Can take Posix-style paths (using "/" as the directory separator) on MacOS X. If the path contains ":" and does not contain "/", it is assumed to be an HFS path.

level — longint: 0 = default, 1 = faster compression, … , 9 = smaller files, but slower compression.

error — longint:

Error	Value
No error	0
Error while opening the original file	-1
Error while creating/opening the compressed file	-2
Error while reading from the original file	-3
Error while writing the compressed file	-4
Error while closing the compressed file	-5

Examples

```
//MacOS
$err := ITK_Mac2gzip("MyHD:MyFolder:Myfile";"MyHD:MyFolder:MyFile.gz") //default compression
//MacOS - through MacBinary
$err := ITK_Mac2Bin ("MyHD:MyFolder:Myfile";"MyHD:MyFolder:Myfile.temp") //convert to MacBinary
$err := ITK_Mac2gzip("MyHD:MyFolder:Myfile.temp";"MyHD:MyFolder:MyFile.gz")
//Windows
$err := ITK_Mac2gzip("C:\\MyHD\\MyFolder\\Myfile";"C:\\MyHD\\MyFolder\\MyFile.gz")
```

ITK_Mac2Hqx

(filePath:T; binhexPath:T) ➞ error:L

Parameter	Type	Description
➞ filePath	text	Pathname of the decompressed file.
➞ binhexPath	text	Pathname of the resulting BinHex file.
➟ error	longint	OS error code.

ITK_Mac2Hqx encodes a Mac or Windows file into BinHex 4.0 format.

ITK V4

filePath, **binhexPath** — text. Can take Posix-style paths (using "/" as the directory separator) on MacOS X. If the path contains ":" and does not contain "/", it is assumed to be an HFS path.

Examples

```
$err := ITK_Mac2Hqx("MyHD:MyFolder:Myfile";"MyHD:MyFolder:MyFile.hqx")  // MacOS
$err := ITK_Mac2Hqx ("C:\\MyHD\\MyFolder\\Myfile";"C:\\MyHD\\MyFolder\\MyFile.hqx")  // Windows
```

ITK_Quoted2Text

(qpText:T; option:L) ➞ text:T

Parameter	Type	Description
➞ qpText	text	Quoted-Printable text to decode.
➞ option	longint	Obsolete.
➟ text	text	Decoded Unicode text.

ITK_Quoted2Text decodes quoted-printable encoded text to Unicode text.

Quoted-printable is a 7-bit format that converts 8-bit characters into an 6-bit encoded representation (e.g. "é" ➞ "=EA"). 7-bit characters are not encoded except the equal sign.

Quoted-printable also deals will lines wrap and text lines to limit their length (wrapped lines are terminated by a space and an equal sign). This command restores wrapped lines into their original state.

ITK V4

Note: ITK_Text2Quoted encodes the text as UTF-8 and then quoted printable encodes that. The reverse is done in ***ITK_Quoted2Text***. Therefore the **option** parameter (where flag = 1 meant “do not apply ISO decoding”) is now ignored. This command can be called with **qpText** as the only parameter, as in the example below.

Example

```
$UnicodeText := ITK_Quoted2Text($myQPtext).
```

ITK_RFC2Secs

(rfcString:T; timeZoneNameFlag:L) ➞ gmtValue:T

Parameter	Type	Description
➞ rfcString	text	RFC formatted date/time string.
➞ timeZoneNameFlag	longint	Use timezone name (if available) instead of an offset.
⬅ gmtValue	text	GMT date/time value.

ITK_RFC2Secs converts an RFC#1123 formatted date and time string (example: Tue, 15 Sept 2015 14:28:51 +0100) into a GMT date/time value.

ITK_RFC2Secs and ITK_Date2Secs always return a GMT based value, so you can compare them directly.

rfcString — text. Be sure to pass RFC#1123 formatted string which are using 4 digits for the year.

timeZoneNameFlag — longint: this parameter was added in v4. If it is non-zero, the timezone name (if available) will be used instead of an offset.

ITK V4

Note: in ITK version 4, the only valid dates are between January 1, 00:00:00, 1970, UTC and January 19, 3:14:07, 2038, UTC, and the values returned by ***ITK_RFC2Secs*** and ***ITK_Date2Secs*** no longer match the values from the old versions.

Examples

Convert an RFC date/time string into the corresponding GMT RFC date/time string:

```
theString := ITK_Secs2RFC (ITK_RFC2Secs (theString);1)
```

Compare two RFC date/time strings, convert them into GMT date/time values:

```
If (ITK_RFC2Secs (myRFCdate1)> ITK_RFC2Secs (myRFCdate2))
   …
End if
```

ITK_Secs2Date

(gmtValue:T; date:D; time:L; timeZoneFlag:L)

Parameter	Type	Description
➞ gmtValue	longint	GMT date / time value.
⬅ date	date	Returned date.
⬅ time	longint	Returned time (as longint).
➞ timeZoneFlag	longint	Indicates in which time zone the date and time are provided.

ITK_Secs2Date converts a GMT date / time value into a date and time (4D formats, time as longint).

You can specify if the resulting date and time must be returned in local time zone or GMT.

timeZoneFlag — longint:

Zone	Value
GMT time zone	0
Local time zone	1

Example

```
 // Get date/time in 4 hours
$dt := ITK_Date2Secs(Current Date; Current Time;1)
$dt := $dt + (4*60*60)  // add 4 hours
ITK_Secs2Date ($dt;$nextDate;$nextTime;0)  // local Time Zone
```

Note: the returned time is typed as a LONGINT. To get the corresponding value in a TIME variable, simply use the following code:

```
C_LONGINT($timeAsLong)
C_TIME($timeAsTime)
ITK_Secs2Date ($dt;$date;$timeAsLong;0)  // local Time Zone
$timeAsTime := $timeAsLong
```

ITK_Secs2RFC

(gmtValue:T; timeZoneFlag:L; timeZoneNameFlag:L) ➞ rfcString:T

Parameter	Type	Description
➞ gmtValue	text	GMT date/time value.
➞ timeZoneFlag	longint	Indicates in which time zone the date and time are provided.
➞ timeZoneNameFlag	longint	Use timezone name (if available) instead of an offset.
⬅ rfcString	text	RFC#1123 formatted string.

ITK_Secs2RFC converts a GMT date/time value into an RFC#1123 formatted date and time string (example: Tue, 15 Sep 2015 14:28:51 +0100).

You can specify if the resulting RFC string must be in GMT time zone or local time zone (according to your computer time zone settings).

timeZoneFlag — longint:

Zone	Value
GMT time zone	0
Local time zone	1

ITK V4

timeZoneNameFlag — longint: this parameter was added in v4. If it is non-zero, the timezone name (if available) will be used instead of an offset. If **timeZoneFlag** is 1 (local time), "GMT" is always used, even if **timeZoneNameFlag** is 0.

Note: only dates between 1 Jan 1970 00:00:00 GMT and 19 Jan 2038 03:14:07 GMT are valid.

Examples

Convert an RFC date/time string into the corresponding GMT RFC date/time string:

```
theString := ITK_Secs2RFC (ITK_RFC2Secs(theString);1)
```

Get the current date and time in GMT time zone:

```
theString := ITK_Secs2RFC (ITK_Date2Secs(Current Date; Current Time;1);1)
```

ITK_Text2B64

(base64text:T; option;L) ➞ result:T

Parameter	Type	Description
➞ base64text	text	Base64 text to decode.
➞ option	longint	1 = add padding if required.
⬅ result	text	Base64 converted text.

ITK_Text2B64 convert a text to Base64. The Base64 conversion is a 8-bit to 6-bit conversion (3 chars encoded into 4 chars). The resulting text will be 33 % larger than the original.

For proper Base64 encoding, please refer to RFC1521.

To comply with MIME, the output stream (encoded bytes) must be represented in lines of no more than 76 characters each. Padding at the end of the data is performed using the “=” character (use **option** value 1).

ITK V4

Note: ***ITK_Text2B64*** encodes the text as UTF-8 and then base64 encodes that. The reverse is done in ITK_B642Text.
If invalid base64 characters are passed in ***ITK_B642Text***, the result is an empty string.

Example

```
$B64pass := ITK_Text2B64("mypassword")
```

ITK_Text2HTML

(text:T; option:L) ➞ HTMLText:T

Parameter	Type	Description
➞ text	text	Text to convert.
➞ option	longint	Conversion option.
⬅ HTMLText	text	HTML converted text.

ITK_Text2HTML converts a text to HTML. By default, all HTML special characters like "<", ">", "&" are converted unless **option** is set to 1. All 8-bit characters are always converted (e.g. é ➞ é).

ITK V4

Note: ***ITK_Text2HTML*** and ITK_HTML2Text will only work with Unicode characters with values 0x0000 to 0xFFFE, which encompasses most modern languages.

option — longint:

Error	Value
Full HTML conversion	0
Do not convert HTML tags ('<', '>' or '"')	1
Extract text (remove HTML tags)	2

Examples

```
$myText := "<B>é</B>"
$HTML := ITK_Text2HTML($myText)  // returns "&lt;B&gt;&eacute;&lt;/B&gt;"
$HTML := ITK_Text2HTML($myText; 1)  // returns "<B>&eacute;</B>"
```

ITK_Text2Quoted

(text:T; option:L) ➞ qpText:T

Parameter	Type	Description
➞ text	text	Text to encode.
➞ option	longint	Obsolete.
⬅ qpText	text	Quoted-Printable encoded text.

ITK_Text2Quoted decodes quoted-printable encoded text.

Quoted-printable is a 7-bit format that converts 8-bit characters into an 6-bit encoded representation (e.g. "é" ➞ "=EA"). 7-bit characters are not encoded except the equal sign.

Quoted-printable also deals will lines wrap and text lines to limit their length (wrapped lines are terminated by a space and an equal sign).

ITK V4

Note: ***ITK_Text2Quoted*** encodes the text as UTF-8 and then quoted printable encodes that. The reverse is done in ITK_Quoted2Text. Therefore the **option** parameter (where flag = 1 meant "do not apply ISO decoding") is now ignored. This command can be called with **text** as the only parameter, as in the example below.

Example

```
$myQPtext := ITK_Text2Quoted($myText).
```

ITK_Text2URL

(text:T; option:L) ➞ URLText:T

Parameter	Type	Description
➞ text	text	Text to convert.
➞ option	longint	Conversion option.
⬅ URLText	text	Converted text.

ITK_Text2URL converts a text to URL encoded text.

option — longint. Two types of URL encoded text are supported by ITK:

Error	Value
Standard URL translation (to use in HTML link for example)	0
Form URL translation (to use in URL encoded form data)	1

ITK V4

Note: ***ITK_Text2URL***/ ITK_URL2Text always converts to/from UTF-8, so the old **option** 4 (do not apply ISO encoding) does nothing. Old **option** 2 was unsafe and is deprecated.

Examples

```
//standard URL translation
$url := ITK_Text2URL("Test 4D+ITK.html")  //returns "Test+4D%2BITK.html"
//form URL translation
$url := ITK_Text2URL("Test 4D+ITK.html";1)  //returns "Test%204D+ITK.html"
```

ITK_Text2uu

(text:T) ➞ uuText:T

Parameter	Type	Description
➞ text	text	Text to encode.
⬅ uuText	text	uuEncoded text.

ITK_Text2uu encodes a text to uuencode format.

uuencoding converts 8-bit binary values to 6-bit ASCII characters.

ITK V4

The text is first converted to UTF-8, which potentially can expands the text up to 300 %, and is then uuencoded, which expands by approximately 38 %. Thus the maximum text length that can be converted is 494MB.

Text longer than that will return an empty string.

Example

```
$uu := ITK_Text2uu("text portion")
```

ITK_URL2Text

(URLText:T; option:L) ➞ text:T

Parameter	Type	Description
➞ URLText	text	URL encoded text to convert.
➞ option	longint	Conversion option.
⬅ text	text	Converted text.

ITK_URL2Text converts URL encoded text to text.

option — longint. Two types of URL encoded text are supported by ITK:

Error	Value
Form URL translation ("+" are translated into " ")	0
Standard URL translation ("+" are not translated)	1

ITK V4

Note: ITK_Text2URL/ ***ITK_URL2Text*** always converts to/from UTF-8, so the old **option** 4 (do not apply ISO decoding) does nothing.

Examples

```
//form URL translation
$text := ITK_URL2Text("Test+4D%2BITK.html")  //returns "Test 4D+ITK.html"
//standard URL translation
$text := ITK_URL2Text("Test+4D%2BITK.html";1)  //returns "Test+4D+ITK.html"
```

ITK_uu2Text

(uuText:T) ➞ text:T

Parameter	Type	Description
➞ uuText	text	uuencoded text to decode.
⬅ text	text	Decoded text.

ITK_uu2Text decodes uuencoded text.

uuencoding converts 8-bit binary values to 6-bit ASCII characters.

ITK V4

The decoded text will be at least 25% smaller. If the original encoded text had non-ASCII characters, the reduction will be greater.

Example

```
$t := ITK_uu2Text($uuText)
```

6 ITK Encryption Commands

About encryption/decryption algorithms

ITK supports several ciphers (encryption/decryption algorithms):

Algorithm name	Algorithm ID	Constant	Key length (bits)
DES	1	kCipherAlgDES	56 (8 x 7-bit chars)
TripleDES	2	kCipherAlgTripleDES	168 (24 x 7-bit chars)
3Way	3	kCipherAlgThreeWay	96
CAST-128 (defined in RFC#2144)	8	kCipherAlgCAST_128	128
TwoFish-128	10	kCipherAlgTWOFISH_128	128
AES-256	25	kCipherAlgAES_256	256
Unix Crypt	28	kCipherAlgUnixCrypt	64

ITK V4

These algorithms work on blocks of data. The following block modes are supported by ITK:

Block mode name	Block mode ID	Constant
CBC	0	kCiphermodeCBC
ECB	1	kCiphermodeECB
CFB	2	kCiphermodeCFB
OFB	3	kCiphermodeOFB
nOFB	4	kCiphermodenOFB

You must use the same algorithm and the same block mode when encrypting and decrypting.

About Digest Calculation commands

A digest is like a “super-checksum” calculated on data. They can be used as digital signatures or to transfer encrypted information

The original information cannot be recovered from the digest.

Uses

- For example, MD5 digests can be used to add a digital signature to emails in order to ensure that the message content has not been modified (see RFC#1544).
- MD5 digests are also used in the POP3 APOP command, which allows user authentication with transferring the user’s password (see RFC#1939).
- MD5 and SHA digests are also used in the SSL protocol.
- RIPEMD-160 digests are used by QuickDNS Pro load balancing system.
- ITK also supports SHA-1 (modified SHA) digests, as well as CRC32 and ADLER32 checksums.

Using ITK digest calculation commands

- The first step is to initialise a digest calculation using ITK_DigestInit.
- Then data is added to the calculation using the ITK_DigestAdd and ITK_DigestBlob commands
- Finally, ITK_DigestCalc is called to get the final digest value (returned as a string).

Commands

ITK_DecryptBlob

(blob:0; secretKey:T; algoID:L; blockMode:L; IV:T; utf8:L) ➞ error:L

Parameter	Type	Description
⇆ blob	blob	Blob to decrypt.
➞ secretKey	text	Key to decrypt the text.
➞ algoID	longint	Type of algorithm to use.
➞ blockMode	longint	Type of block mode to use.
➞ IV	text	Initial Vectors.
ITK V4 ➞ utf8	longint	Convert **secretKey** and **IV** from UTF-8.

ITK_DecryptBlob decrypts a text using you choice of cipher algorithm and block mode.

ITK V4 **utf8** — longint. If not passed or zero, the text is converted from Mac Roman before being decrypted. This ensures backward compatibility if you are storing encrypted blobs. If the **utf8** parameter is non-zero, the text is converted from UTF-8.

Note: we recommend you pass **utf8** = 1 to ensure the full range of Unicode characters can be represented in the **secretKey** and **IV**. In other words, if you want to use characters that are not in the Mac Roman character set (in the **secretKey** or **IV**), then you must pass 1 for **utf8**.

If you want to use **utf8** = 1, you should first convert blobs stored in the database to UTF-8:

```
C_BLOB($blob)
While(Not(End selection([table])))
   $blob:=[table]blob
   ITK_DecryptBlob($blob;$key;$algo;$mode;$iv)
   ITK_EncryptBlob($blob; $key; $algo; $mode;$iv;1)
   [table]blob:=$blob
   SAVE RECORD([table])
   NEXT RECORD([table])
End while
```

error — longint:

Error	Value
No error	0
Could not initialize cipher	-1
Unknown algorithm	-2

See also About encryption / decryption algorithms.

Example

```
err := ITK_DecryptBlob (myBlob;"12345678";kCipherAlgDES;kCiphermodeECB)
```

ITK_DecryptText

(blob:O; secretKey:T; algoID:L; blockMode:L; IV:T; text:T; utf8:L) ➞ error:L

Parameter	Type	Description
➞ blob	blob	Blobed text to decrypt.
➞ secretKey	text	Key to decrypt the text.
➞ algoID	longint	Type of algorithm to use.
➞ blockMode	longint	Type of block mode to use.
➞ IV	text	Initial Vectors.
⬅ text	text	Decrypted text.
➞ utf8	longint	Convert text from UTF-8.

ITK V4

ITK_DecryptText decrypts a text using your choice of cipher algorithm and block mode.

ITK V4

Note: calls to this command need to be updated in ITK version 4.

Because encryption results in raw data which cannot safely be stored in a Unicode string, the encrypted text must be stored in a blob.

blob contains the encrypted text, the decrypted **text** is returned.

utf8 — longint. If passed and non-zero, indicates that **secretKey** and **IV** should be converted to UTF-8 before 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 decrypted **text**, **secretKey** and **IV**. In other words, if you want to use characters that are not in the Mac Roman character set (in the **text**, **secretKey** or **IV**), then you must pass 1 for **utf8**.

error — longint:

Error	Value
No error	0
Could not initialize cipher	-1
Unknown algorithm	-2

See also About encryption / decryption algorithms.

Example

```
err := ITK_DecryptText(text;"12345678";kCipherAlgDES;kCiphermodeECB)
```

ITK_DigestAdd

(digestRef:L; text:T; utf8:L)

Parameter	Type	Description
➞ digestRef	longint	Digest reference.
➞ text	text	Text to add to the calculation.
➞ utf8	longint	Convert text to UTF-8.

ITK V4

ITK_DigestAdd adds **text** to a current digest calculation referenced by **digestRef**.

This command may be called several times during a digest calculation.

ITK V4

utf8 — longint. 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.

See also About Digest Calculation commands.

Examples

```
$digestRef := ITK_DigestInit(2)  // start MD2 digest calculation
ITK_DigestAdd($digestRef;"Here is some text")
ITK_DigestAdd($digestRef;"and some more")
ITK_DigestBlob($digestRef;myBlob)
$myDigest := ITK_DigestCalc($digestRef)  // get the result
```

ITK_DigestBlob

(digestRef:L; blob:O)

Parameter	Type	Description
➞ digestRef	longint	Digest reference.
➞ blob	blob	Data to add to the calculation.

ITK_DigestBlob adds data contained in a **blob** to a current digest calculation referenced by **digestRef**.

This command may be called several times during a digest calculation.

See also About Digest Calculation commands.

Example

See the example above for ITK_DigestAdd

ITK_DigestCalc

(digestRef:L; digestFormat:L) ➞ digest:T

Parameter	Type	Description
➞ digestRef	longint	Digest reference.
➞ digestFormat	longint	1 = lowercase hexadecimal encoded string.
⬅ digest	text	Digest calculation result.

ITK_DigestCalc terminates the digest calculation and returns the calculated **digest**.

After calling this command, the **digestRef** previously returned by ITK_DigestInit is no longer valid.

ITK V4

Note: 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.

See also About Digest Calculation commands.

Example

See the example above for ITK_DigestAdd

ITK_DigestInit

(digestType:L) ➞ digestRef:L

Parameter	Type	Description
➞ digestType	longint	Digest type.
⬅ digestRef	longint	Digest reference.

ITK_DigestInit initializes a new digest calculation and returns a digest reference, which will be needed in the next step of a digest calculation (ITK_DigestAdd or ITK_DigestBlob, then ITK_DigestCalc).

digestType — longint:

Digest type	Value
SHA digests (conforms to FIPS PUB 180)	1
MD2 digests (conforms to RFC#1319)	2
MD4 digests (conforms to RFC#1320)	4
MD5 digests (conforms to RFC#1321)	5
SHA-1 digests (conforms to FIPS PUB 180-1)	9
RIPEMD160 digests	10
CRC32 checksum (conforms to ISO3309)	100
ADLER32 checksum	101

See also About Digest Calculation commands.

Example

See the example above for ITK_DigestAdd

ITK_EncryptBlob

(blob:0; secretKey:T; algoID:L; blockMode:L; IV:T; utf8:L) ➞ error:L

Parameter	Type	Description
➞ blob	blob	Blob to encrypt.
➞ secretKey	text	Key to encrypt the text.
➞ algoID	longint	Type of algorithm to use.
➞ blockMode	longint	Type of block mode to use.
➞ IV	text	Initial Vectors.
➞ utf8	longint	Convert **secretKey** and **IV** to UTF-8.

ITK_EncryptBlob encrypts a **blob** content using you choice of cipher algorithm and block mode.

You will be able to decrypt it using ITK_DecryptBlob.

utf8 — longint. If not passed or zero, the text is converted to Mac Roman before being decrypted. This ensures backward compatibility if you are storing encrypted blobs. If the **utf8** parameter is non-zero, the text is converted from UTF-8.

Note: we recommend you pass **utf8** = 1 to ensure the full range of Unicode characters can be represented in the **secretKey** and **IV**. In other words, if you want to use characters that are not in the Mac Roman character set (in the **secretKey** or **IV**), then you must pass 1 for **utf8**.

error — longint:

Error	Value
No error	0
Could not initialize cipher	-1
Unknown algorithm	-2

See also About encryption / decryption algorithms.

Example

```
err := ITK_EncryptBlob(myBlob;"12345678";kCipherAlgDES;kCiphermodeECB)
```

ITK_EncryptText

(text:T; secretKey:T; algoID:L; blockMode:L; IV:T; text:T; utf8:L) ➞ error:L

Parameter	Type	Description
➞ text	text	Clear text to encrypt.
➞ secretKey	text	Key to encrypt the text.
➞ algoID	longint	Type of algorithm to use.
➞ blockMode	longint	Type of block mode to use.
➞ IV	text	Initial Vectors.
⬅ blob	blob	Blob containing the encrypted text.
➞ utf8	longint	Convert text from UTF-8.

ITK_EncryptText encrypts a **text** using your choice of cipher algorithm and block mode.

ITK V4

Note: calls to this command need to be updated in ITK version 4.

Because encryption results in raw data which cannot safely be stored in a Unicode string, the encrypted text must be stored in a blob.

You will be able to decrypt it using ITK_DecryptText.

text is the text to encrypt.

blob contains the returned encrypted text.

utf8 — longint. If passed and non-zero, indicates that **text**, **secretKey** and **IV** should be converted to UTF-8 before encryption. 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**, **secretKey** and **IV**. In other words, if you want to use characters that are not in the Mac Roman character set (in the **text**, **secretKey** or **IV**), then you must pass 1 for **utf8**.

error — longint:

Error	Value
No error	0
Could not initialize cipher	-1
Unknown algorithm	-2

See also About encryption / decryption algorithms.

Example

```
err := ITK_EncryptText(text;"12345678";kCipherAlgDES;kCiphermodeECB)
```

ITK_Rot13Blob

(blob:O)

Parameter	Type	Description
⇆ blob	blob	Original blob, encrypted blob returned.

ITK_Rot13Blob applies a ROT13 “light encryption” to a **blob**.

ROT13 is an email standard to hide data. It is not a real encryption, only a shift in characters to make the data unreadable.

ROT13 is auto-reversible, apply it a second time to get the original data back.

If the original data was 7-bit only, the resulting data will still be a 7-bit only text.

The original blob is directly “encrypted”.

Example

```
ITK_Rot13Blob(myBlob)
```

ITK_Rot13Text

(originalText:T) ➞ encryptedText:T

Parameter	Type	Description
➞ originalText	text	Original text.
← encryptedText	text	Encrypted text.

ITK_Rot13Text applies a ROT13 “light encryption” to a **blob**.

ROT13 is an email standard to hide text. It is not a real encryption, only a shift in characters to make the text unreadable.

ROT13 is auto-reversible, apply it a second time to get the original text back.

In other words ROT13(ROT13(text)) = text

If the original text was 7-bit only, the resulting text will still be a 7-bit only text.

The original blob is directly “encrypted”.

Example

```
$rot13 := ITK_Rot13Text("Hello")
```

ITK SSL Commands

About ITK SSL support

ITK supports SSL (Secure Sockets Layer) versions 2 and 3 as well as TLS (Transport Layer Security) version 1.

This support is mostly transparent: all TCP-based ITK code should work in SSL with almost no modification.

Here are the changes you need to do in order to use SSL instead of TCP:

Acting as a **server**:

■ call ITK_SSLSetCert after calling ITK_TCPListen .

This will let you speficy the certificate/private key that will be used for this SSL stream.

Acting as a **client**:

■ pass the option value 1024 in ITK_TCPOpen **tcpOpt** parameter (4th parameter).

This will switch this stream to the SSL “client”.

All other ITK TCP commands can be used on an SSL stream.

SSL Server Certificates and private keys

In order **to act as a server, you must have a certificate** that will allow the client to identify your server.

A certificate is a text file looking like this:

```
-----BEGIN CERTIFICATE-----
MIICqzCCAhSgAwIBAgIDKyXqMA0GCSqGSIb3DQEBBAUAMIGHMQswCQYDVQQGEwJa
QTEiMCAGA1UECBMZRk9SIFRFU1RJTkcgUFVSUE9TRVMgT05MWTEdMBsGA1UEChMU
…
jhsh9nP5wIYjia5GxuN4HWAJpnSFwC98LZcMqLU09+6XMymX8R2Voj0vBI8NIfQk
HgHQIPQuIXcgaNONSP5A
-----END CERTIFICATE-----
```

A certificate is linked to a private key, which is needed in order to use the certificate. The certificate is useless without the private key. The private key is also a text file looking like this:

```
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQDBO/ieW0VhcH/SniA6r9EPls6KBGWZDQVhzWLURXLe00VKgDOk
RrZ+L8umdlBZM1y1RPWaCM3U56WiOUfq/2wflnbtjgb14m7lbtr0e0UuKDZa7SJJ
…
Yvv+effpcMccy+amOxkCQGKGsjoMKhjgoifv1AElDWsFCK3OWiUFEWqWDYJcDVXK
RBQBN/B6lJof5D4GN1L3B4gHgcsxkSSqYj1ud34zppk=
-----END RSA PRIVATE KEY-----
```

A private key can be password protected: in that case, you'll need to provide the password in order to use the private key. A password protected private key looks like this:

```
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,D4A75236509E8F8C
Ii6hR/SwPlUgDUfy7bIk//Sc52zhL76ecvSKO4PBJ1rYpOYnHUrQF0YEvDuVuD8j
JdUGaC9i1bCKFU2Z3ryTXFUmU4rvWyXFs0Xq/sYb+o3kfwRo96VnpR7b1OsGom3K
…
y5WwsxBhUpKZOtgPjgGNCcIAf4dkBmzM50HrkSPW0kICRisZiuDyWYmMXL/dYDAh
7ez5I1K/GJSaAJyJxW7rzNpLa+V1gEEwDSXjoNNxXWJbmcuHRkf04A==
-----END RSA PRIVATE KEY-----
```

There are countless online guides to generating certificates.

It is highly recommended to always try with test certificates before buying a certificate from a certification authority. This will ensure that the certificate format is compatible with ITK.

Note about certificate and private key file format:

ITK uses certificates in PEM format.

These files are text files. Windows and MacOS are using differents kind of "end of lines" in text files. Windows standard is ending lines with CR+LF, while MacOS uses only a CR.

ITK recognizes CRLF end of lines under both MacOS and Windows, and also recognizes CR under MacOs (but not under Windows).

Be sure to use the right kind of "end of line" in your certificate and private key files, otherwise these files will be unusable.

Chained certificates

A certificate chain file must contain **all** of the certificates in the chain, starting from the user certificate and proceeding to the root.

ITK uses the openssl call SSL_CTX_use_certificate_chain_file, documented here.

Here's from the documentation:

SSL_CTX_use_certificate_chain_file() loads a certificate chain from file into ctx. The certificates must be in PEM format and must be sorted starting with the subject's certificate (actual client or server certificate), followed by intermediate CA certificates if applicable, and ending at the highest level (root) CA.

Commands

ITK_SSLGetError

(streamRef:L; errString:T) ➞ errNum:L

Parameter	Type	Description
➞ streamRef	longint	Returned by ITK_TCPOpen, or ITK_TCPListen.
← errString	text	Error text.

ITK_SSLGetError provides details on the last error that occurred on the given stream.

An error code is returned, as well as a error string including details about the error.

errString — text. The returned string is formatted as follows:

- error followed by the numeric error code in hexadecimal
- the part of the SSL library where the error occurred
- the routine name where the error occurred
- a text explanation of the error

Example: "error:0906A068:PEM routines:PEM_do_header:bad password read".

The error 0906A068 occurred in the PEM routines, more precisely in the PEM_do_header routine, and the problem was a "bad password read".

Example

```
$streamRef := ITK_TCPListen(0;0;443)  // listen on HTTPS port
If ($streamRef # 0)
    // set our server certificate
   $err := ITK_SSLSetCert($streamRef;"myHD:myFolder:keyout.pem"; "myHD:myFolder:cert.pem";"")
   If ($err # 0)
      $err := ITK_SSLGetError($streamRef; $error)
      ALERT("SSL "+$error)
      …
```

ITK_SSLSetCert

(streamRef:L; certPath:T; pKeyPath:T; password:T; chainCertPath:T) ➞ error:L

Parameter	Type	Description
➞ streamRef	longint	Returned by ITK_TCPOpen, or ITK_TCPListen.
➞ certPath	text	Full pathname of the certificate.
➞ pKeyPath	text	If left to 0, will set default private key, certificate and password.
➞ password	text	Optional password for the private key.
➞ chainCertPath	text	Optional full pathname of the chained certificate (loaded if not empty).

ITK_SSLSetCert allows to specify a private key and a certificate for a given stream.

Note: the private key and certificate files must be in PEM format.

ITK_SSLSetCert must be used **before** the stream gets connected . It is recommended to call ***ITK_SSLSetCert*** just after calling ITK_TCPListen and before calling ITK_TCPStatus or ITK_TCPWaitConn.

Be sure to use full pathnames for the private key and the certificate files.

If **streamRef** is set to 0, ***ITK_SSLSetCert*** will allow to set a default private key file, a default certificate file and a default password. These make sure that a certificate is set before the stream gets connected.

See the example below.

Since ITK v3.0.2, ***ITK_SSLSetCert*** accepts a fifth optional parameter: the path to a chained certificate (in PEM format).

ITK V4

certPath, **pKeyPath** — text. Can take Posix-style paths (using "/" as the directory separator) on MacOS X. If the path contains ":" and does not contain "/", it is assumed to be an HFS path.

Compatibility note: both ITK v3 & v4 support exchanging **certPath** and **pKeyPath** parameters.
If setting **certPath** fails (using PEM and then ASN1 format), then **pKeyPath** is tried.
If setting **pKeyPath** fails (using PEM and then ASN1 format), **certPath** is tried.

In other words, both following orders work:
- streamRef, certPath, pKeyPath, password, chainCertPath
- streamRef, pKeyPath, certPath, password, chainCertPath

error — longint:

Error	Value
No error	0
Invalid **streamRef**	-1
Not an SSL stream	-2
Invalid certificate	-3
Invalid private key or password	-4

Examples

```
$streamRef := ITK_TCPListen(0;0;443)  //listen on HTTPS port
If ($streamRef # 0)
    //set our server certificate immediately
   $err := ITK_SSLSetCert($streamRef;"myHD:myFolder:keyout.pem"; "myHD:myFolder:cert.pem";"")
   …
Using default values mechanism:
$err := ITK_SSLSetCert($streamRef;"myHD:myFolder:keyout.pem"; "myHD:myFolder:cert.pem";"")
…
$streamRef := ITK_TCPListen(0;0;443)  //listen on HTTPS port
If ($streamRef # 0)
    //no need to set the certificate infos, we can use the streamRef immediately
   $s := ITK_TCPWaitConn($s)
```

ITK_SSLSetCiphers

(streamRef:L; cipherList:T) ➞ error:L

Parameter	Type	Description
➞ streamRef	longint	Returned by ITK_TCPOpen, or ITK_TCPListen.
➞ cipherList	text	Cipher list definition. Leave blank to use default the cipher list.

ITK_SSLSetCiphers allows to select which ciphers will be accepted on the given SSL stream. The cipher list follows OpenSSL cipher list syntax.

error — longint. 0 or OpenSSL error code.

Example

```
$streamRef := ITK_TCPListen(0;0;443)  //listen on HTTPS port
If ($streamRef # 0)
  //set our server certificate
$err := ITK_SSLSetCiphers($streamRef;"")
…
```

ITK_SSLStrmInfo

(streamRef:L; sslVers:T; cipher:T; keyBits:L; certData:T) ➞ error:L

Parameter	Type	Description
➞ streamRef	longint	Returned by ITK_TCPOpen, or ITK_TCPListen.
➞ sslVers	text	SSL version (as string).
➞ cipher	text	Name of the cipher used.
➞ keyLen	longint	Length (in bits) of the cipher key.
← certData	text	Certificate data.

ITK_SSLStrmInfo gets information about the SSL stream like the version of SSL or TLS used, the cipher (encryption algorithm) used, and the length (in bits) of the key used to encrypt the transmission as well as some data about the remote certificate (if you're acting as a client).

Note: this command can only return valid information when the stream is connected and SSL negociation has occurred.

sslVers — text. Possible values:

- "SSLv2"
- "SSLv3"
- "TLSv1"

keyLen — longint. Common values:

- 40 (export grade)
- 56 (DES for example)
- 128 (strong encryption)
- 168 (TripleDES for example)

error — longint:

Error	Value
No error	0
Invalid **streamRef**	-1
Not an SSL stream	-2
The stream is not connected	-3

Example

```
$streamRef := ITK_TCPListen(0;0;443)  //listen on HTTPS port
If ($streamRef # 0)
    //set our server certificate
   $err := ITK_SSLSetCert($streamRef;"myHD:myFolder:keyout.pem"; "myHD:myFolder:cert.pem";"")
    //wait for connection
   $err := ITK_TCPWaitConn($streamRef)
    //get info about the connection
   $err := ITK_SSLStrmInfo($streamRef; $sslVers; $cipher; $keyLen)
   If ($keyLen >= 128)
       //we are in "strong" 128bits (or above) mode
      ....
```

ITK TCP/IP Commands

TCP/IP commands include low-level and high-level tools dealing with the TCP/IP protocol.

Low-level TCP/IP commands are used to open connections (streams) between two computers using the TCP/IP protocol to reliably transfer data.

High level TCP/IP commands provided by ITK allow more sophisticated uses of TCP streams.

ITK high level TCP/IP commands do not directly interact with each other.

For example, ITK_TCPSendFile and ITK_TCPRecvFile are not “symetric” commands that can be used as is on both ends of a stream to transfer a file.

ITK_TCPSendFile sends the data contained in a file, while ***ITK_TCPRecvFile*** stores all incoming data into a file, but the incoming data has no link with its source (a file or anything else) and ***ITK_TCPRecvfile*** will need to meet an “end of file” condition to stop storing data in the receiving file. You must handle this yourself.

This also applies for blob sending and receiving.

To determine when the “end” condition occurs, you may need to send the size of the file or blob you’re about to transfer so that the receiver knows the number of bytes to receive.

You may also use an “end string”, however make sure this end string is not contained in the data you’re about to transfer.

You may also choose to close the stream on the sender side.

Commands

ITK_SetTimeout

(streamRef:L; newTimeoutTicks:L) ➞ newTimeoutSecs:L

Parameter	Type	Description
➞ streamRef	longint	Returned by ITK_TCPOpen, or ITK_TCPListen.
➞ newTimeoutTicks	longint	Timeout value in ticks (1/60 s) or 0 = no timeout (default).
⬅ newTimeoutSecs	longint	New timeout value in seconds.

ITK_SetTimeout sets the default timeout used by all "send" ITK commands (ITK_TCPSend, ITK_TCPSendFile, ITK_TCPSendPict, ITK_TCPSendBlob).

By default, ITK doesn't use any timeout when sending data. It can be useful to set a timeout in order to avoid possible "send locks" when the remote host becomes unreachable (e.g. crash or disconnection).

When the timeout is reached in one of the ***ITK_TCPSendXXX*** commands, the commands will return control, but the status of the stream will remain the same.

The best place to set the timeout for a stream is just after a successful call to ITK_TCPOpen or ITK_TCPListen.

Example

```
$stream := ITK_TCPOpen("mail.e-node.net";25)
if ($stream#0)
  //all ITK_TCPSendXXX calls will have a 2 mn timeout
    $curTimeout := ITK_SetTimeout($stream;2*60*60)
    …
```

ITK_TCPChRcv

(streamRef:L) ➞ result:L

Parameter	Type	Description
➞ streamRef	longint	Returned by ITK_TCPOpen, or ITK_TCPListen.
⬅ result	longint	Number of bytes (positive values) or error code -1 = invalid stream reference number.

ITK_TCPChRcv returns the number of bytes available in the receive buffer of a TCP stream.

Note: the number of bytes received after calling ITK_TCPRcv may be greater than the value returned by ***ITK_TCPChRcv*** as data may have been received between the two calls.

Example

```
If (ITK_TCPChRcv($stream)#0)  // do we have any data available?
   $result := ITK_TCPRcv($stream;$buff)
   …
```

ITK_TCPClose

(streamRef:L) ➞ error:L

Parameter	Type	Description
➞ streamRef	longint	Returned by ITK_TCPOpen, or ITK_TCPListen.
⬅ error	longint	0 (no error) or error code -1 = invalid stream reference number.

ITK_TCPClose sends the data remaining in the output buffer (see ITK_TCPSend) and tells the remote host that no more data will be sent.

After closing a stream, you can still receive data on it until you call ITK_TCPRelease. This is possible because a TCP stream is made of two half streams, each one used to send data from one host to the other. Each half-stream can be closed separately (also called half-close).

Note: closing a stream does not release the stream, but only tells the remote host that you're done sending data.

Example

```
$stream := ITK_TCPOpen("http://www.e-node.net/itk";80)
If ($stream>0)
   …
   $err := ITK_TCPClose($stream)
   …
   $err := ITK_TCPRelease($stream)
End if
```

ITK_TCPGetStrm

(index:L) ➞ result:L

Parameter	Type	Description
➞ index	longint	Index value.
⬅ result	longint	StreamRef (if index > 0). Number of streams (if index=0). 0 (index out of range).

ITK_TCPGetStrm retrieves the list of all current TCP streams allocated by ITK.

index — longint:

- Pass 0 to get the current number of streams.
- Pass 1…n to get the nth stream.

result — longint:

- If index > 0, returns the StreamRef.
- If index = 0, returns the numbert of allocated streams.
- If the returned value is 0, the index is out of range.

Example

```
$nbStrm := ITK_TCPGetStrm(0)  // get the number of allocated streams
For ($i;1;$nbStrm)
   $status := ITK_TCPStatus(ITK_TCPGetStrm($i))
End for
```

ITK_TCPGlobInfo

(infoSelector:L) ➞ result:L

Parameter	Type	Description
➞ infoSelector	longint	Global info selector.

ITK_TCPGlobInfo returns global information about the TCP layers.

infoSelector — longint:

- Pass 0 to get the current number of streams.
- Pass 1…n to get the nth stream.

result — longint:

Selector	Value
Maximum TCP connections	5
Number of connection attempts	6
Number of connections opened	7
Number of connections accepted	8
Number of closed connections	9
Number of aborted connections	10
Bytes received (with SSL streams, the number of "raw" bytes is used)	11
Bytes sent (with SSL streams, the number of "raw" bytes is used)	12

Example

```
maxTCPstrm := ITK_TCPGlobInfo(5)  // maximum number of streams
```

ITK_TCPInfos

(localIP:L; version1:L; version2:L; ITKversion:L) ➞ error:L

Parameter	Type	Description
← localIP	longint	Local IP address.
← version1	longint	Version (platform) of TCP/IP layers.
← version2	longint	Obsolete, always returns -1.
← ITKversion	longint	ITK's version number.

ITK_TCPInfos returns global information about the TCP/IP layers.

localIP — longint. On multihomed systems, the primary (or default) local IP address is returned.

version1 — longint:

Platform	Value
MacOS	4
Windows	5

ITKversion — longint. $MMmrssbb (in hexadecimal) where:

- MM = major version number
- m = minor version number
- r = release version number
- ss = stage code ($20 = dev, $40 = alpha, $60 = beta, $80 = Final)
- bb = build number

Example

```
$err := ITK_TCPInfos(locAddr;TCPvers;OTvers)
If (TCPvers=5)  // we're on Windows
    // do something
End if
```

ITK_TCPListen

(hostFilter:L; remotePort:I; localPort:I; rcvBuffer:L; tcpOpt:L; oldStream:L; localIP:L) ➞ streamRef:L

Parameter	Type	Description
➞ hostFilter	longint	TCP address of remote host.
➞ remotePort	integer	TCP remote port.
➞ localPort	integer	Local port number.
➞ rcvBuffer	longint	Receive buffer size for this stream.
➞ tcpOpt	longint	TCP options.
➞ oldStream	longint	Obsolete, pass 0.
➞ localIP	longint	Local IP address (multihoming).
⬅ streamRef	longint	Stream reference number.

ITK_TCPListen opens a “passive” or “listening” TCP stream to receive calls.

Note: all streams opened using ***ITK_TCPListen*** MUST be released using ITK_TCPRelease.

As with ITK_TCPOpen, control is returned immediately without waiting for a connection to be received and established.

Use ITK_TCPStatus or ITK_TCPWaitConn to wait for the connection to be established.

Note: under MacOS X, your application needs to run with root privileges in order to listen on ports ranging from 1 to 1023.

hostFilter — longint. Pass 0 to accept connections from any remote host, otherwise pass a numeric IP address to only accept connections from the specified remote host.

remotePort — longint. Pass 0 to accept connections from any remote port numbers, otherwise pass a port number to only accept connections from the specified port number (e.g. 80 for HTTP, 25 for SMTP, see RFC#1700 for assigned numbers).

localPort — longint. Pass the local port number on which you want to listen for incoming connections (e.g. 80 for HTTP, 25 for SMTP, see RFC#1700 for assigned numbers). If **localPort** is set to 443 (HTTPS standard port number), ITK will switch automatically to SSL.

rcvBuffer — longint. Accepts values between 8192 and 32767.Pass 0 for ITK to use a default buffer of 8KB (8192 bytes).

tcpOpt — longint. Set to kITKSSLListenStream (2048) to create an SSL “server” stream instead of a TCP stream.

localIP — longint. Pass 0 if you want to accept incoming connections on all local IP addresses, otherwise, pass one of the local IP addresses to only accept incoming connections on the specified IP address.

streamRef — longint. Will return 0 if the stream couldn’t be created.

Examples

```
  //create a TCP listening stream on HTTP port
$stream := ITK_TCPListen(0;0;80)  //listen on HTTP port number
If ($stream#0)
   …
   $err := ITK_TCPRelease($stream)
End if
  //create an SSL listening stream on HTTPS port
$stream := ITK_TCPListen(0;0;443;0;kITKSSLListenStream)  //listen on HTTPS port number
If ($stream#0)
   …
   $err := ITK_TCPRelease($stream)
End if
```

ITK_TCPOpen

(hostName:T; remotePort:I; rcvBuffer:L; tcpOpt:L; oldStream:L; localPort:I; localIP:L) ➞ streamRef:L

Parameter	Type	Description
➞ hostName	text	TCP address of remote host (URL or IP address).
➞ remotePort	integer	TCP remote port.
➞ rcvBuffer	longint	Receive buffer size for this stream.
➞ tcpOpt	longint	TCP option.
➞ oldStream	longint	Obsolete, pass 0.
➞ localPort	integer	Local port number.
➞ localIP	longint	Local IP address (multihoming).
⬅ streamRef	longint	Stream reference number.

ITK_TCPOpen opens a TCP stream asynchronously (returns control immediately without waiting for the connection to be established with the remote host).

Note: all streams opened using ***ITK_TCPOpen*** MUST be released using ITK_TCPRelease.

As with ITK_TCPListen, control is returned immediately without waiting for a connection to be received and established.

Use ITK_TCPStatus or ITK_TCPWaitConn to wait for the connection to be established.

hostName — text. "http://www.e-node.net" or "212.27.40.241".

remotePort — longint. Pass 0 to accept connections from any remote port numbers, otherwise pass a port number to only accept connections from the specified port number (e.g. 80 for HTTP, 25 for SMTP, see RFC#1700 for assigned numbers). If **remotePort** is set to 443 (HTTPS standard port number), ITK will switch automatically to SSL.

rcvBuffer — longint. Accepts values between 8192 and 32767.Pass 0 for ITK to use a default buffer of 8KB (8192 bytes).

tcpOpt — longint. Set to kITKSSLClientStream (1024) to create an SSL “server” stream instead of a TCP stream.

localPort — longint. Pass the local port number on which you want to listen for incoming connections (e.g. 80 for HTTP, 25 for SMTP, see RFC#1700 for assigned numbers).

localIP — longint. Pass 0 if you want to accept incoming connections on all local IP addresses, otherwise, pass one of the local IP addresses to only accept incoming connections on the specified IP address.

streamRef — longint. Will return 0 if the stream couldn’t be created.

Example

```
  // open a stream with an HTTP server
$stream := ITK_TCPOpen("www.e-node.net";80)
If ($stream#0)
    …
   $err := ITK_TCPRelease($stream)
End if
  // open an SSL stream with an HTTPS server
$stream := ITK_TCPOpen("www.e-node.net";443;0;kITKSSLClientStream)  // SSL stream
If ($stream#0)
   …
   $err := ITK_TCPRelease($stream)
End if
```

ITK_TCPRcv

(streamRef:L; dataStorage:T; maxLen:L; filterFlag:L; option:L; endString:T; timeout:L) ➞ result:L

Parameter	Type	Description
➞ streamRef	longint	Returned by ITK_TCPOpen, or ITK_TCPListen.
➞ dataStorage	text	Storage variable or field.
➞ maxLen	longint	Maximum length of data to be received.
➞ filterFlag	longint	Encoding flag(s).
➞ option	longint	Option for **dataStorage**.
➞ endString	text	End receive string.
➞ timeout	longint	Maximum time to wait for **endString** (in ticks).
⬅ result	longint	Number of bytes (positive values) or error code -1 = invalid stream reference number.

ITK_TCPRcv receives data through an established TCP stream.

Remaining data in ITK's send buffer is sent if present.

Filtering can be applied on the received text.

dataStorage — text. Content is replaced or appended depending on **option**.

maxLen — longint. Pass 0 if you want to receive all available data (up to 2GB, which is the limit for Unicode text fields or variables).

filterFlag — longint. One or more encodings can be applied before sending the data (or putting it into ITK's receive buffer). You may combine (add) flag values to apply several filters at once.

ITK V4

Note: if no "from" encoding conversion is specified in the **filterFlag** parameter, it defaults to UTF-8. If a "to" encoding conversion is passed in the **filterFlag** parameter, kITKInvalidConversion (-7) is returned.

option — longint:

Option	Value	Constant
Do not append received data to previous data present in **dataStorage**	0	
Append received data to previous data present in **dataStorage**	1	kITKAppendFlag
The **endString** will remain in ITK's receive buffer	2	kITKKeepEndString
Do not search **endString** in original data when kITKAppendFlag is used	4	kITKSearchNewData

endString — text. If this string is found in incoming data or accumulated data (when kITKAppendFlag is used), ***ITK_TCPRcv*** will return all data up to this string (including it), remaining data will be kept in ITK's receive buffer.

timeout — longint. Timeout in ticks (1/60 s) or special values:

Timeout	Value
Infinite time	-1
Return control immediately	0

ITK_TCPRcv will return control immediately when the timeout is set to 0.

When the timeout is not 0, ***ITK_TCPRcv*** will return control when one of these conditions is valid:

- **endString** (if not empty) is received (and according to the **option** flag)
- **maxlen** bytes have been received (if **maxlen** is not 0)
- data have been received and no **maxlen** is specified (or left to 0)
- **timeout** is over
- the length of **dataStorage** is greater or equal to 2 GB.

Note: the number of bytes received after calling ***ITK_TCPRcv*** may be greater than the value returned by ITK_TCPChRcv as data may have been received between the two calls.

Example

```
$stream := ITK_TCPOpen("mail.e-node.net";25)
if ($stream#0)
   … // wait for connection to be established
   $rcvdLen := ITK_TCPRcv($stream; $data)
   …
```

In the following example, the first call to ***ITK_TCPRcv*** will receive data up to a pair of CR/LF (<>CRLF) during a maximum of 2 minutes.

Then the second call to ***ITK_TCPRcv*** will append received data into http_req until a <>CRLF is received in new data or $len data is accumulated in http_req during a maximum of 2 minutes.

```
 // Receive HTTP header
$err := ITK_TCPRcv($c;http_req;0;0;<>CRLF+<>CRLF; 2*60*60)
…
If (http_req = "POST @")
   $err := ITK_TCPRcv($c;http_req;$len;1+4;<>CRLF;2*60*60)  // receive "POST" data
   …
```

ITK_TCPRecvBlob

(streamRef:L; dataStorage:O; maxLen:L; filterFlag:L; option:L; endString:T; timeout:L) ➞ error:L

Parameter	Type	Description
➞ streamRef	longint	Returned by ITK_TCPOpen, or ITK_TCPListen.
➞ dataStorage	blob	Storage variable or field.
➞ maxLen	longint	Maximum length of data to be received.
➞ filterFlag	longint	Encoding flag(s).
➞ option	longint	Option for **dataStorage**.
➞ endString	text	End receive string.
➞ timeout	longint	Maximum time to wait for **endString** (in ticks).
⬅ error	longint	0 (no error) or error code -1 = invalid stream reference number.

ITK_TCPRecvBlob receives data through an established TCP stream.

Remaining data in ITK's send buffer is sent if present.

Filtering can be applied on the received text.

dataStorage — text. Content is replaced or appended depending on **option**.

maxLen — longint. Pass 0 if you want to receive all available data.

filterFlag — longint. One or more encodings can be applied before sending the data (or putting it into ITK's receive buffer). You may combine (add) flag values to apply several filters at once.

ITK V4

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 **filterFlag** 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.

Note: 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.

option — longint:

Option	Value	Constant
Append to existing blob	1	kITKAppendFlag
Do not release the stream after receiving the blob	2	
Look for **endString** only in new incoming data	4	kITKSearchNewData
Do not search **endstring** in original data when kITKAppendFlag is used	8	

endString — text. ***ITK_TCPRecvBlob*** will return data until this string is found. If this string is found in incoming data, ***ITK_TCPRecvBlob*** will return all data up to this string, remaining data will be kept in ITK's receive buffer.

ITK V4

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 **filterFlag**.

timeout — longint. Timeout in ticks (1/60 s) or special values:

Timeout	Value
Infinite time	-1
Return control immediately	0

Note: the stream will be released automatically after receiving the blob, unless option 2 is used.

ITK_TCPRecvBlob will return control:

- immediately if **endString** is empty
- when **endString** is received
- when **maxLen** bytes have been received
- after **timeout** has expired

Example

```
$stream := ITK_TCPOpen("mail.e-node.net";25)
If ($stream#0)
   … // wait for connection to be established
   $error := ITK_TCPRecvBlob($stream; $myBlob; 0; 0; 2) // receive available data in a blob
   …
```

ITK_TCPRecvFile

(streamRef:L; pathname:T; filterFlag:L; option:L; timeout:L; maxLen:L) ➞ error:L

Parameter	Type	Description
➞ streamRef	longint	Returned by ITK_TCPOpen, or ITK_TCPListen.
➞ pathname	text	Pathname of the file where to store incoming data.
➞ filterFlag	longint	Encoding flag(s).
➞ option	longint	Option for **dataStorage**.
➞ timeout	longint	Timeout value in seconds.
➞ maxLen	longint	Maximum length of data to be received.
⬅ error	longint	0 (no error) or error code -1 = invalid stream reference number.

ITK_TCPRecvFile stores all data received on a TCP stream into a file.

Filtering can be applied on the received file text.

ITK V4

pathname — text. ***ITK_TCPRecvFile*** and ITK_TCPSendFile can take Posix-style paths (using "/" as the directory separator) on MacOS X. If the path contains ":" and does not contain "/", it is assumed to be an HFS path.

filterFlag — longint. One or more encodings can be applied before sending the data (or putting it into ITK's receive buffer). You may combine (add) flag values to apply several filters at once.

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 **filterFlag** 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.

option — longint:

Option	Value	Constant
Append to existing file	1	kITKAppendFlag
Do not release the stream after receiving the file	2	

timeout — longint. Timeout in ticks (1/60 s) or 0 = infinite time.

Note: the stream is automatically released before returning control to 4D, unless option 2 is used.

ITK_TCPRcvFile returns control when:

- the stream is closed by the remote host
- no data has been received during the **timeout** value
- the maximum length has been reached
- an error occurred on the stream or about the file (I/O error, disk full, etc.)

maxLen — longint. Pass 0 if you want to receive all available data.

ITK V4

Note: when filtering is applied with ***ITK_TCPRcvFile***, the entire file is loaded into memory and then filtered. At least two copies of the file are held in memory briefly during filtering. Keep this in mind if you need to worry about available memory.

Example

```
$stream := ITK_TCPOpen("host.domain.com";80)
If ($stream#0)
   …  // wait for connection to be established
   $err := ITK_TCPRecvFile($stream;"C:\\MYFILE.TXT") `
End if
```

ITK_TCPRelease

(streamRef:L; option:L) ➞ error:L

Parameter	Type	Description
➞ streamRef	longint	Returned by ITK_TCPOpen, or ITK_TCPListen.
➞ option	longint	Optional flag.
⬅ error	longint	0 (no error) or error code -1 = invalid stream reference number.

ITK_TCPRelease releases a TCP stream and all its associated buffers.

After this call, the **streamRef** should no longer be used.

Note: when calling ***ITK_TCPRelease*** , all data remaining to be sent will not be sent. To avoid this, you must call ITK_TCPClose and then monitor ITK_TCPStatus to check that the remote host has received all remaining data and that you closed your half-stream.

Each successfull call to ITK_TCPOpen and ITK_TCPListen must be balanced by a call to ***ITK_TCPRelease***.

option — longint. 2 = force internal listener to stop listening.

Example

```
$stream := ITK_TCPOpen("http://www.e-node.net";80)
If ($stream#0)
   …
   $err := ITK_TCPRelease($stream)
End if
```

ITK_TCPSend

(streamRef:L; data:T; flushFlag:I; filterFlag:L) ➞ result:L

Parameter	Type	Description
➞ streamRef	longint	Returned by ITK_TCPOpen, or ITK_TCPListen.
➞ data	text	Data to send.
➞ flushFlag	integer	Output buffer settings.
➞ filterFlag	longint	Encoding flag(s).
⬅ result	longint	Number of bytes (positive values) or error code -1 = invalid stream reference number (it may have been released by the remote host).

ITK_TCPSend sends data through an established TCP stream. ITK has its own send buffer which helps reducing the number of calls to lower level TCP/IP layers when small chunks of data are sent.

Filtering can be applied on the sent text.

Note: 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.

flushFlag — longint:

Flag	Value	Constant
Don't use ITK output buffer, flush its content if it wasn't empty	0	kITKFlush
Use ITK's output buffer, will be flushed regularly by ITK	1	kITKBuffer
Maximum number of bytes that may be buffered If the output buffer contains more than the value passed, ITK will flush it	>255	

Note: ITK's send buffer will also be flushed when calling ITK_TCPRcv or ITK_TCPClose.

filterFlag — longint. One or more encodings can be applied before sending the data (or putting it into ITK's send buffer). You may combine (add) flag values to apply several filters at once.

ITK V4

Note: if no "to" encoding conversion is specified in the **filterFlag** parameter, it defaults to UTF-8. If a "from" encoding conversion is passed in the **filterFlag** parameter, kITKInvalidConversion (-7) is returned.

Example

```
$stream := ITK_TCPOpen("http://www.e-node.net/itk";80)
If ($stream#0)
   … // wait for connection to be established
    // use ITK's send buffer
   $err := ITK_TCPSend($stream;"GET / HTTP/1.0"+$crlf;1)
    // Flush ITK's send buffer
   $err := ITK_TCPSend($stream;"User-Agent: myWebBrowser/1.0"+$crlf+$crlf)
   …
```

ITK_TCPSendBlob

(streamRef:L; blob:O; flushFlag:L; filterFlag:L; startOffset:L; endOffset:L) ➞ error:L

Parameter	Type	Description
➞ streamRef	longint	Returned by ITK_TCPOpen, or ITK_TCPListen.
➞ blob	blob	Data to send.
➞ flushFlag	longint	Obsolete, ignored.
➞ filterFlag	longint	Encoding flag(s).
➞ startOffset	longint	Starting offset of data to send.
➞ endOffset	longint	Ending offset of data to send.
➞ error	longint	0 (no error) or error code -1 = invalid stream reference number.

ITK_TCPSendBlob sends a blob content or a part of it through an established TCP stream.

ITK has its own send buffer which help reducing the number of calls to lower level TCP/IP layers when small chunks of data are sent.

Filtering can be applied on the sent data.

Note: ITK_SetTimeout can be used to set a “maximum send timeout” to be used by ITK when sending data.
In the event that the remote host doesn't close the stream gracefully (disconnection, crash, etc.) this “maximum send timeout” will prevent ITK from continuously trying to send data.

filterFlag — longint. One or more encodings can be applied before sending the data (or putting it into ITK’s send buffer). You may combine (add) flag values to apply several filters at once.

ITK V4

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 **filterFlag** 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.

Note: 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.

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)
```

startOffset — longint. Pass 0 to send data from the beginning of the blob.

endOffset — longint. Pass 0 to send data up to the end of the blob.

Example

```
$stream := ITK_TCPOpen("www.www.e-node.net";80)
If ($stream#0)
   … //wait for connection to be established
   $err := ITK_TCPSendBlob($stream;myBlob) //send the whole blob content
   $err := ITK_TCPSendBlob($stream;myBlob;0;0;0;1024) //send first 1024 bytes of the blob
   …
```

ITK_TCPSendFile

(streamRef:L; pathname:T; filterFlag:L; sendBlockSize:L; startOffset:L; endOffset:L; option:L) ➞ error:L

Parameter	Type	Description
➞ streamRef	longint	Returned by ITK_TCPOpen, or ITK_TCPListen.
➞ pathname	text	Pathname of the file to send.
➞ filterFlag	longint	Encoding flag(s).
➞ sendBlockSize	longint	Send block size.
➞ startOffset	longint	Starting offset of the portion of the file to send.
➞ endOffset	longint	Ending offset of the portion of the file to send.
➞ option	longint	MacOS resource/data forks option.
← error	longint	0 (no error) or error code -1 = invalid stream reference number (it may have been released by the remote host), or -2 = error opening the file.

ITK_TCPSendFile sends a file through a TCP stream. After sending the file, the stream is left opened.

Filtering can be applied on the sent file.

The last two parameters allow to send only a portion of the file.

ITK's internal buffer is flushed before sending the file.

Note: ITK_SetTimeout can be used to set a "maximum send timeout" to be used by ITK when sending data.
In the event that the remote host doesn't close the stream gracefully (disconnection, crash, etc) this "maximum send timeout" will prevent ITK from continuously trying to send data.

ITK V4

pathname — text. ITK_TCPRecvFile and ***ITK_TCPSendFile*** can take Posix-style paths (using "/" as the directory separator) on MacOS X. If the path contains ":" and does not contain "/", it is assumed to be an HFS path.

filterFlag — longint. One or more encodings can be applied before sending the data (or putting it into ITK's send buffer). You may combine (add) flag values to apply several filters at once.

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 **filterFlag** 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 **filterFlag** parameter.

ITK V4

Note: 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 sent.

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)
```

option — longint:

Option	Value	Constant
Send data fork	0	
Send resource fork (MacOS only)	1	

Note: when filtering is applied with ***ITK_TCPSendFile***, the entire file is loaded into memory and then filtered. At least two copies of the file are held in memory briefly during filtering. Keep this in mind if you need to worry about available memory.

Example

```
$stream := ITK_TCPOpen("host.domain.com";80)
If ($stream#0)
 … // wait for connection to be established
 $err := ITK_TCPSendFile($stream;"C:\\MYFILE.TXT")
 …
```

ITK_TCPSendPict

(streamRef:L; picture:P; startOffset:L; endOffset:L) ➞ error:L

Parameter	Type	Description
➞ streamRef	longint	Returned by ITK_TCPOpen, or ITK_TCPListen.
➞ picture	picture	Picture to send.
➞ startOffset	longint	Starting offset of data to send.
➞ endOffset	longint	Ending offset of data to send.
⬅ error	longint	0 (no error) or error code -1 = invalid stream reference number (it may have been released by the remote host).

ITK_TCPSendPict sends a picture content or a part of it through an established TCP stream. ***ITK_TCPSendPict*** will look for GIF, PNG or JPEG data and only send them if present in the picture (unless the **endOffset** parameter is set to -1, see below).

Note: ITK_SetTimeout can be used to set a “maximum send timeout” to be used by ITK when sending data.
In the event that the remote host doesn't close the stream gracefully (disconnection, crash, etc.) this “maximum send timeout” will prevent ITK from continuously trying to send data.

startOffset — longint. Pass 0 to send data from the beginning of the picture.

endOffset — longint. Pass -1 to send the full content of the picture without looking for GIF, PNG or JPEG data.

Example

```
COMPRESS PICTURE($myPict;"jpeg";500)
$err :=ITK_TCPSendPict($stream;$myPict)
```

or, using the ITK_Pict2GIF replacement 4D project method:

```
$err :=ITK_TCPSendPict($stream;ITK_Pict2GIF($myPict))
```

ITK_TCPStatus

(streamRef:L; selector:L) ➞ status:L

Parameter	Type	Description
➞ streamRef	longint	Returned by ITK_TCPOpen, or ITK_TCPListen, or global info selector.
➞ selector	longint	Type of information to be returned.
⬅ status	longint	Current stream status code or error code -1 = invalid stream reference number.

ITK_TCPStatus returns the current status of a TCP stream or miscellaneous information about the stream.

Use it to wait for a connection to be established (status=8) or for the remote host to acknowledge a half-close.

streamRef — longint. Value -1 means that a global information selector is passed instead of the stream reference number:

Global info selector value	Value	Constant
Maximum TCP connections	-5	kITKMaxTCPConnections
Number of connection attempts	-6	kITKConnAttempts
Number of connections opened	-7	kITKConnOpened
Number of connections accepted	-8	kITKConnAccepted
Number of closed connections	-9	kITKConnClosed
Number of aborted connections	-10	kITKConnAborted
Bytes received	-11	kITKBytesReceived
Bytes sent	-12	kITKBytesSent

selector — longint:

Selector value	Value	Constant
Current status of **streamRef**	0	kITKStreamStatus
Bytes received	14	kITKRetransmittedBytes
Bytes sent	17	
Raw bytes of encrypted data received	23	
Raw bytes of encrypted data sent	24	

status — longint (see RFC#793, page 21):

ITK V4

Status value	Value	Constant
Invalid **streamRef** error	-1	kStreamStatusInvalid
No connection exists on this stream	0	kStreamStatusClosed
Listening for an incoming connexion	2	kStreamStatusListen
Incoming connection is being established	4	kStreamStatusSYNReceived
Outgoing connection is being established	6	kStreamStatusSYNSent
Connection is up	8	kStreamStatusEstablished
Connection is up; close has been sent	10	kStreamStatusFINWait1
Connection is up; close has been sent and acknowledged	12	kStreamStatusFINWait2
Connection is up; close has been received	14	kStreamStatusCloseWait
Connection is up; close has been issued and received	16	kStreamStatusClosing
Connection is up; close has been issued and received	18	kStreamStatusLastACK
Connection is broken	20	kStreamStatusTimeWait

Example

```
//wait for the connection to be established
Repeat
   $s := ITK_TCPStatus($stream)
Until (($s>=8) | ($s<0)) //end if connected or error
```

ITK_TCPStatus2A

(arrStreams:Y; arrStatus:Y; minStatus:L) ➞ index:L

Parameter	Type	Description
➞ arrStreams	array	Longint array containing stream reference numbers returned by ITK_TCPOpen, or ITK_TCPListen.
➞ arrStatus	array	Longint array containing status values.
➞ minStatus	longint	Status value to look for.
⬅ index	longint	Item number of the first stream that reaches minStatus or error code -1 = invalid stream reference number.

ITK_TCPStatus2A returns the current status of an array of streams and the index of the first stream which status is greater than or equal to **minStatus**, or -1 (error on the stream).

This command can be used to monitor the status of a list of streams and wait for a connection or a close to be acknowledged.

status — longint (see RFC#793, page 21):

ITK V4

Status value	Value	Constant
Invalid stream reference error	-1	kStreamStatusInvalid
No connection exists on this stream	0	kStreamStatusClosed
Listening for an incoming connexion	2	kStreamStatusListen
Incoming connection is being established	4	kStreamStatusSYNReceived
Outgoing connection is being established	6	kStreamStatusSYNSent
Connection is up	8	kStreamStatusEstablished
Connection is up; close has been sent	10	kStreamStatusFINWait1
Connection is up; close has been sent and acknowledged	12	kStreamStatusFINWait2
Connection is up; close has been received	14	kStreamStatusCloseWait
Connection is up; close has been issued and received	16	kStreamStatusClosing
Connection is up; close has been issued and received	18	kStreamStatusLastACK
Connection is broken	20	kStreamStatusTimeWait

Example

```
ARRAY LONGINT($streams;10)
ARRAY LONGINT($status;10)
For ($i;1;10)
   $streams{$i} := ITK_TCPListen(0;0;80)
End for
 // wait for the connection to be established on a list of streams
Repeat
   $i := ITK_TCPStatus2A($streams; $status; 8)
   If ($i>0)
      $s := ITK_TCPStatus($streams{$i})
      Case of
      :($s >= 8)  // stream connected
         …  // handle the HTTP hit
      :($s<0)  // error on stream
         $err := ITK_TCPRelease($streams{$i})
         $streams{$i} := ITK_TCPListen(0;0;80)
      …
      End case
   End if
Until (quit)
```

ITK_TCPStrmInfo

(streamRef:L; remoteIP:L; remotePort:I; localPort:I; obsolete:L; localIP:L) ➞ error:L

Parameter	Type	Description
➞ streamRef	longint	Returned by ITK_TCPOpen, or ITK_TCPListen.
⬅ remoteIP	longint	IP address of the remote host.
⬅ remotePort	integer	Port number used by the remote host.
⬅ localPort	integer	Local port number.
⬅ obsolete	longint	Will always return -1.
⬅ localIP	longint	Local IP address.
⬅ error	longint	0 (no error) or error code -1 = invalid stream reference number.

ITK V4

ITK_TCPStrmInfo returns information about an established TCP stream.

localIP — longint. This value is useful for multihomed systems. On non multihomed system, the local IP address is always returned.

Examples

```
$err := ITK_TCPStrmInfo($stream;remAddr;remPort;locPort;srtt;locAddr)
ALERT("Remote host address:"+ITK_Addr2Name(remAddr;2))
```

Note: **localPort** and **remotePort** may return negative values, because 4D is using signed integers. To get the original positive port number, you can use the code below.

```
C_LONGINT(locPort; remPort)  // MUST be typed as LONGINTS !!!
$err := ITK_TCPStrmInfo ($stream;remAddr;remPort;locPort;srtt;locAddr)
If (locPort < 0)
   locPort := locPort + 65536
End if
If (remPort < 0)
   remPort := remPort + 65536
End if
```

ITK_TCPUnRcv

(streamRef:L; data:T; option:L; filterFlag:L) ➞ error:L

Parameter	Type	Description
➞ streamRef	longint	Returned by ITK_TCPOpen, or ITK_TCPListen.
➞ data	text	Data to put back in ITK's receive buffer.
➞ option	longint	Optional flag.
➞ filterFlag	longint	Encoding used in the unreceive buffer.
⬅ error	longint	0 (no error) or error code -1 = invalid stream reference number.

ITK V4

ITK_TCPUnRcv puts previously received data back into ITK's receive buffer.

This data will be received in the following call to receiving commands, e.g. ITK_TCPRcv, ITK_TCPRcvBlob, ITK_TCPRcvFile.

option — longint:

Option	Value	Constant
Put data at the end of the "unreceive buffer"	0	kITKAppendUnreceiveData
Put data at the begionning of the "unreceive buffer"	1	kITKInsertUnreceiveData

filterFlag — longint. 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***.

ITK V4

Example

```
$result := ITK_TCPRcv($stream; $data)  // receive some data
If ($data # "+OK@")
   $err := ITK_TCPUnRcv($stream;$data)  // put it back in ITK's buffer to receive it again
End if
```

ITK_TCPWaitConn

(streamRef:L; minStatus:L; timeout:L) ➞ result:L

Parameter	Type	Description
➞ streamRef	longint	Returned by ITK_TCPOpen, or ITK_TCPListen.
➞ minStatus	longint	Minimum status value to wait for.
➞ timeout	longint	Maximum time to wait (in seconds) or 0 = wait indefinitely.
⬅ result	longint	Stream's status or error error code -1 = invalid stream reference number.

ITK_TCPWaitConn waits for a TCP stream to be in a certain status or to return an error (-1).

This command can be used to wait for a stream to establish connection. It is synchronous, control will be returned:

- when the stream status is greater or equal to **minStatus**,
- when **timeout** expires,
- when an error occurs with the stream.

Example

```
$stream := ITK_TCPOpen("mail.e-node.net";25)
 //wait for connection to be established
$status := ITK_TCPWaitConn($stream;8;10)  //wait 10 seconds
If ($status=8)
   …
End if
$err := ITK_TCPRelease($stream)
```

ITK UDP Commands

UDP is a connectionless protocol that can be used to transfer small amounts of data without requiring to open a connection between two computers.

UDP is used for example by the DNS query protocol or by Syslog.

Commands

ITK_UDPCreate

(startPort:L; obsolete:L; bufferSize:L) ➞ result:L

Parameter	Type	Description
➞ startPort	longint	Port number of the UDP endpoint to create.
➞ obsolete	longint	Obsolete, ignored.
➞ bufferSize	longint	The size of the receiving buffer will determine how many received datagrams can be buffered.
← result	longint	UDP endpoint reference number, or 0 if the endpoint couldn't be created.

ITK_UDPCreate creates an UDP endpoint on the specified port or ports in order to receive or send UDP datagrams.

This command returns a reference to the UDP endpoint which you later use in further UDP calls.

Each UDP endpoint created using ***ITK_UDPCreate*** must be released using ITK_UDPRelease (see example below).

Example

```
$udp := ITK_UDPCreate(4000;0;8192)
If ($udp # 0)
   Repeat
      $err := ITK_UDPRcv($udp; $data; $remAddr; $remPort)
      …
   Until …
   $err := ITK_UDPRelease($udp)
   $udp := 0  // so that we're sure we won't release it again
End if
```

ITK_UDPMTUSize

(remAddr:L) ➞ result:L

ITK V4

Note: this command is kept for compatibility reasons, although it never did anything else than returning 548 in all previous ITK versions. In version 4 it returns 576 bytes, which is the absolute minimum MTU that any possible host might have.The **remAddr** parameter is (and has always been) ignored.

ITK_UDPRcv

(udpRef:L; data:T; remAddr:L; remPort:L; obsolete:L; timeout:L) ➞ error:L

Parameter	Type	Description
➞ udpRef	longint	UDP endpoint reference number returned by ITK_UDPCreate.
➞ data	text	Received data.
➞ remAddr	longint	IP address of the sender host.
➞ remPort	longint	Port number of the sender's UDP endpoint.
➞ obsolete	longint	Obsolete, ignored.
➞ timeout	longint	Maximum time to wait for a datagram.
➞ error	longint	0 (no error) or error code -1.

ITK_UDPRcv receives an UDP packet on an UDP endpoint.

ITK V4

Note: text is converted from UTF-8 when received.

data — text. Be aware that you can receive an empty datagram. In that case, you can obtain information about the sender through the **remAddr** and **remPort** parameters.

timeout — longint:

Timeout	Range
Timeout in ticks	<0
Infinite time	0
Timeout in seconds	>0

Example

See example above for ITK_UDPCreate.

ITK_UDPRelease

(udpRef:L) ➞ error:L

Parameter	Type	Description
➞ udpRef	longint	UDP endpoint reference number returned by ITK_UDPCreate.
➟ error	longint	0 (no error) or error code -1.

ITK_UDPRelease releases an UDP endpoint. The **udpRef** should not be used anymore.

Make sure to call ***ITK_UDPRelease*** only once for each **udpRef**.

Example

See example above for ITK_UDPCreate**.**

ITK_UDPSend

(udpRef:L; data:T; remAddr:L; remPort:L; obsolete:L) ➞ error:L

Parameter	Type	Description
➞ udpRef	longint	UDP endpoint reference number returned by ITK_UDPCreate.
➞ data	text	Data to send.
➞ remAddr	longint	IP address of the recipient host.
➞ remPort	longint	Destination port number on the remote host.
➞ obsolete	longint	Obsolete, ignored.
⬅ error	longint	0 (no error) or error code -1.

ITK_UDPSend sends data through an UDP endpoint to another UDP endpoint.

ITK V4

Note: text is converted to UTF-8 when sent.

data — text. Be aware that you can receive an empty datagram. In that case, you can obtain information about the sender through the **remAddr** and **remPort** parameters.

timeout — longint:

Timeout	Range
Timeout in ticks	<0
Infinite time	0
Timeout in seconds	>0

Example

```
$udp := ITK_UDPCreate(4000;0;8192)
If ($udp # 0)
   $data := ITK_UDPSend($udp; $data; $remAddr; $remPort)
   $err := ITK_UDPRelease($udp)
   $udp := 0  //so that we're sure we won't release it again
End if
```

ITK Utility Commands

About ITK Interprocess Communication commands

ITK's IPC commands (ITK_NbIPCMsg, ITK_SendIPCMsg, ITK_RcvIPCMsg, ITK_ResetIPCMsg) can be used to exchange data between processes through communication channels (thus avoiding to read and write 4D variables, semaphores, etc.). These channels are memory based to provide speed efficient communications.

The number of channels and the number of messages queued in a channel is limited to 2 billion and by the available memory.

Commands

ITK_Addr2Name

(hostAddr:L; option:L) ➞ result:L

Parameter	Type	Description
➞ hostAddr	longint	As returned for example by ITK_Name2Addr.
➞ option	longint	Option values.
⬅ result	longint	Depending on **option** value.

ITK_Addr2Name translates an IP address (as a long integer) into its corresponding "network name" (string).

option — longint. Option flag to specify the desired result format:

Option	Value	Constant
Return the full address if possible otherwise return a "dotted address"	0	kAddressResolutionAny
Return only full text addresses obtained from a DNS or an empty string if no answer could be received from the DNS	1	kAddressResolutionName
Always return a "dotted address"	2	kAddressResolutionDotted

Examples

```
$addr := ITK_Name2Addr("www.e-node.net")
$name := ITK_Addr2Name($addr)  // should return something like "server.something.com"
$dotted := ITK_Addr2Name($addr;2)  // should return "82.165.15.6"
```

ITK_Name2Addr

(hostName:T; obsolete:L) ➞ hostAddr:L

Parameter	Type	Description
➞ hostName	text	Host name to resolve.
➞ obsolete	longint	Obsolete, ignored.
⬅ hostAddr	longint	IP address or 0 = no address returned, couldn't resolve **hostName**.

ITK_Name2Addr translates a hostname into its corresponding IP address. If the original address is empty, this command returns the local IP address.

Examples

```
$addr := ITK_Name2Addr("www.e-node.net")
$addr := ITK_Name2Addr("www.e-node.net";-1)  //use DNS load balancing (random choice)
$addr := ITK_Name2Addr("www.e-node.net";2)  //get second address (may return 0)
```

ITK_NbIPCMsg

(ipcChannel:L) ➞ nbMsg:L

Parameter	Type	Description
➞ ipcChannel	longint	IPC channel number.
⬅ nbMsg	longint	Number of messages stored in this IPC channel.

ITK_NbIPCMsg returns the number of messages currently stored on the specified IPC channel.

Note: IPC channel numbers are used to create as many IPC channels as you need. For example, you can use the **Current Process** function in 4D language to allocate one IPC channel for each 4D process or use any other channel numbering.

Example

```
While (ITK_NbIPCMsg(1)>0)
  //process IPC messages
  …
End while
```

ITK_OpenFile

(pathname:T) ➞ fileRef:L

Parameter	Type	Description
➞ pathname	text	Pathname of the file to open.
← fileRef	longint	File reference of the opened file.

ITK_OpenFile opens a file in read-only mode.

Note: the variable used to store the **fileRef** value must be typed as **C_TIME** when compiled.

fileRef — longint. 0 if the file could not be opened. The returned value can be used in 4D file related commands like **RECEIVE PACKET** , **CLOSE DOCUMENT**, etc. It is equivalent to the one returned by **Open Document**.

ITK V4

Note: this command has been kept for compatibility reasons.
It is recommended to directly use **Open document**($path, Read Mode) instead.

Example

```
$file := ITK_OpenFile("C:\\MyFile.TXT")
If ($file#0)
   RECEIVE PACKET($file;$data;32000)
   …
   CLOSE DOCUMENT($file)
End if
```

ITK_PictRead

(filePath:T; startOffset:L; endOffset:L; option:L) ➞ picture:P

Parameter	Type	Description
➞ filePath	text	Pathname of picture file to read.
➞ startOffset	longint	Starting offset. Pass 0 to start reading at the beginning of the file.
➞ endOffset	longint	End offset. Pass 0 to read up to the end of the file.
➞ option	longint	PICT format option (deprecated).
ITK V4 ➞ picture	picture	Picture read from file. If an error occurred, this picture will have its size set to 0.

ITK_PictRead reads a picture file (PICT, GIF, PNG or JFIF/JPEG) and returns it as a 4D picture.

ITK V4 Note: the PNG format is supported.

ITK V4 **filePath** — text. Can take Posix-style paths (using "/" as the directory separator) on MacOS X. If the path contains ":" and does not contain "/", it is assumed to be an HFS path.

option — longint. 0 = default behaviour, 1 = remove the 512 header bytes in PICT files.

Note: this parameter is still honored, but should be considered deprecated since PICT files are pretty useless these days.

Example

```
$pict := ITK_PictRead("MyHD:MyFolder:MyPict")
$err := ITK_TCPSendPict($stream; $pict)
```

ITK_PictSave

(picture:P; filePath:T; startOffset:L; endOffset:L; option:L) ➞ error

Parameter	Type	Description
➞ picture	picture	Picture to save.
➞ filePath	text	Pathname of picture file.
➞ startOffset	longint	Starting offset. 0 = save from the beginning of the picture.
➞ endOffset	longint	End offset. 0 = save up to the end of the picture.
➞ option	longint	PICT format option (deprecated).
⬅ error	longint	0 or OS error code.

ITK V4

ITK_PictSave saves the content of a picture field or variable into a file.

ITK V4

Notes: 4D may store multiple representations of a picture in a single picture variable. Only the first representation in the picture is saved.

The PNG format is supported.

filePath — text. Can take Posix-style paths (using "/" as the directory separator) on MacOS X. If the path contains ":" and does not contain "/", it is assumed to be an HFS path.

ITK V4

Note: because MacOS X does not recommend using file type / creator, the appropriate filename extension is added to the path if necessary.

option — longint. 0 = default behaviour, 1 = add the 512 header bytes in PICT files.

Note: this parameter is still honored, but should be considered deprecated since PICT files are pretty useless these days.

Example

```
$err := ITK_PictSave($pict;"MyHD:MyFolder:MyPict")
```

ITK_PictSize

(picture:P; obsolete:L; obsolete:L; bottom:L; right:L; type:L) ➡ pictSize

Parameter	Type	Description
➡ picture	picture	Picture.
⬅ obsolete	longint	Obsolete, returns 0.
⬅ obsolete	longint	Obsolete, returns 0.
⬅ bottom	longint	Bottom coordinate of the picture.
⬅ right	longint	Right coordinate of the picture.
⬅ type	longint	Picture type.
⬅ pictSize	picture	Size (in bytes) of the picture.

ITK_PictSize extracts size and other information about a **picture**.

ITK V4

Note: 4D may store multiple representations of a picture in a single picture variable. This command always returns info on the first representation.

type — longint:

Type	Value
Mac style PICT	0
GIF	1
JPEG/JFIF	2
Mac PICT containing JPEG data (deprecated)	3
PNG	4

ITK V4

Example

```
$size := ITK_PictSize($pict;$top;$left;$bottom;$right;$type)
$height := $bottom-$top
$width := $right-$left
```

ITK_RcvIPCMsg

(ipcChannel:L; option:L) ➞ message:T

Parameter	Type	Description
➞ ipcChannel	longint	IPC channel number.
➞ option	longint	0 = default behaviour, 1 = keep message in channel queue.
⬅ message	text	Received message.

ITK_RcvIPCMsg reads a message from the specified IPC channel. If no message is available, returns an empty string.

Note: IPC channel numbers are used to create as many IPC channels as you need. For example, you can use the **Current Process** function in 4D language to allocate one IPC channel for each 4D process or use any other channel numbering.

option — longint. This parameter can be used to keep the message in the IPC channel queue. In that case, the same message will be returned on the next call to ***ITK_RcvIPCMsg***.

Example

```
$msg := ITK_RcvIPCMsg(1;0)
If ($msg = "quit")
   QUIT 4D
End if
```

ITK_Register

(registrationCode; options; email) ➞ result

Parameter	Type	Description
➞ registrationCode	text	Pass the registration key to register your copy of ITK. The key is either linked to the 4D or 4D Server serial number (individual licenses), to the machine ID (merged licenses), to the name of the company/developer (unlimited annual licenses) or to the product (master keys for Online registration).
➞ options	longint	An optional longint combining up to 4 bits: force check, Online registration options.
➞ email	text	Online registration option: developer email to notify when a license is issued or resent.
⬅ result	longint	0 or error code.

ITK_Register is used to register the ITK plugin for standalone or server use.

Please see the License Types section for detailed information about the licensing options available for ITK.

Multiple calls to ***ITK_Register*** are allowed. The plugin will be activated if at least one valid key is used, and all subsequent calls to ***ITK_Register*** will return 0, unless the force check bit is set to true in the **options** parameter.

registrationCode — You must call ***ITK_Register*** with a valid registration key, otherwise ITK will operate in demonstration mode - it will cease to function after 20 minutes. In case a master key is used the plugin will attempt a connection to e-Node's license server for Online registration.

options — Optional. This parameter combines up to 4 bits as described below. The default mode (**registrationCode** being a passed as the only parameter) is silent: no force check, no confirmation, no alert, no email.

Bit number	Description
0	Force check: if this bit is is on (true), **registrationCode** is tested regardless of current registration state. If the plugin was not previously registered and the result is 0, it is registered the same way as if the bit was off (or the whole **options** parameter omitted) If the plugin was previously successfully registered, a registration error will be returned in result in case **registrationCode** is invalid, but the plugin will remain registered.
1	Online registration option: confirm connection "Is it OK to connect to e-Node's license server to register ITK?"
2	Online registration option: display alert if registration error
3	Online registration option: display alert if registered

email — Optional. The developer email address where to send Online registration information.

result — 0 or error code:

Result code	Description
0	OK
1	Beta license has expired
2	Invalid license
3	The license has expired
4	The OEM license has expired
5	The maximum number of users has been exceeded
6	The license is for a different environment (e.g. the licence is for a single-user version, but you are using it with 4D Server)
7	The license is linked to a different 4D license
8	Invalid merged license
9	Only serial/ID licenses are allowed in text license files (includes Register button and Online registration)
10	Unauthorized master key (Online registration)
11	Can't connect to e-Node's license server to perform Online registration
12	No Online registration license available for this master key (unknown or all used)

When ***ITK_Register*** is called with an empty string, the license dialog will be displayed if ITK is not registered and the dialog was not yet displayed. This allows you to show the registration dialog to your users without effectively calling a ITK command or displaying a ITK area.

Note: alternately to ***ITK_Register***, you can place a plain text file into your 4D Licenses folder or use the Demo mode dialog "Register" button. This is only valid for non-unlimited licenses.

Basic example

```
C_LONGINT ($result)
$result:=ITK_Register ("YourRegistrationKey")
Case of
   :($result=2)
      ALERT ("The ITK licence is invalid.")
   :($result=3)
      ALERT ("The ITK licence has expired.")
   etc.
End case
```

Example with multiple calls

```
C_LONGINT ($result)  //ignored in this case
$result:=ITK_Register ("Registration key one")
$result:=ITK_Register ("Registration key two")
$result:=ITK_Register ("Registration key three")
```

etc.

```
If ($result#0)  //registration failed on all keys
   ALERT ("ITK could not be registered.")
End if
```

Force check example

In this example we assume that only "Registration key two" is valid, but you want to check the other keys status.

```
C_LONGINT ($result)
$result:=ITK_Register ("Registration key one";1)  //invalid, will return an error, the plugin isn't registered
$result:=ITK_Register ("Registration key two";1)  //valid, will return 0, the plugin is registered
$result:=ITK_Register ("Registration key three";1)  //invalid, will return an error, the plugin is still registered
```

Online registration examples

ITK V4

Confirm connection, alert if successful, alert if failed, send email notification to developer@4dchampions.com:

```
C_LONGINT ($result)
$result:=ITK_Register ("Master key";0 ?+1 ?+2 ?+3;" developer@4dchampions.com")
```

Silent connection, alert if successful, alert if failed, no email notification:

```
C_LONGINT ($result)
$result:=ITK_Register ("Master key";0 ?+2 ?+3)
```

ITK_ResetIPCMsg

(ipcChannel:L)

Parameter	Type	Description
→ ipcChannel	longint	IPC channel number.

ITK_ResetIPCMsg removes all messages stored on the specified IPC channel.

Note: IPC channel numbers are used to create as many IPC channels as you need. For example, you can use the **Current Process** function in 4D language to allocate one IPC channel for each 4D process or use any other channel numbering.

Example

ITK_ResetIPCMsg(1)

ITK_SendIPCMsg

(ipcChannel:L; message:T)

Parameter	Type	Description
→ ipcChannel	longint	IPC channel number.
→ message	text	Message to send on the above channel.

ITK_SendIPCMsg sends a message on the specified IPC channel.

The message will be stored in the channel message queue.

Note: IPC channel numbers are used to create as many IPC channels as you need. For example, you can use the **Current Process** function in 4D language to allocate one IPC channel for each 4D process or use any other channel numbering.

Example

ITK_SendIPCMsg(1;"quit")

Copyrights and Trademarks

ITK is exclusively published worldwide by e-Node.

4th Dimension, 4D and 4D Server are trademarks of 4D SAS.

Windows, Excel and Vista are trademarks of Microsoft Corporation.

Macintosh, MacOS and MacOS X are trademarks of Apple, Inc.

ITK V4

ITK version 4 was written by Aparajita Fishman, based on the original work by Christian Quest.

Index

Symboles

A

B

C

D

E

F

G

H

I

L

M

O

P

Q

R

S

T

U

V

W

X