



# TCP Deux

## Developer Documentation v1.0.0b01

©1998-2001 Deep Sky Technologies, Inc. All Rights Reserved.  
Published and Distributed Worldwide by Deep Sky Technologies, Inc.

Deep Sky Technologies, Inc.  
P.O. Box 6897  
Vero Beach, FL 32961-6897  
(561) 794-9494



<http://www.deepskytech.com/>

### Software Engineers

James T. Crate  
Robert T. McGoye  
Steven G. Willis

### Manual

Steven G. Willis



# Software License and Limited Warranty

Please read this license carefully before using the software. By using the software, you agree to become bound by the terms of this agreement, which includes the software license and warranty disclaimer (collectively referred to herein as the "agreement"). This agreement constitutes the complete agreement between you and Deep Sky Technologies, Inc. If you do not agree to the terms of this agreement, do not use the software and promptly destroy all copies in your possession, physical and electronically.

**1. Ownership of Software:** The enclosed manual and computer programs ("Software") were developed and are copyrighted by Deep Sky Technologies, Inc. ("DSTi") and are licensed, not sold, to you by DSTi for use under the following terms, and DSTi reserves any rights not expressly granted to you. DSTi retains ownership of all copies of the Software itself. Neither the manual nor the Software may be copied in whole or in part except as explicitly stated below.

**2. License:** DSTi, as Licensor, grants to you, the Licensee, a non-exclusive, non-transferable right to use this Software subject to the terms of the license as described below:

- a. You may make backup copies of the Software for your use provided they bear the DSTi copyright notice.
- b. You may use this Software in an unlimited number of distributed copies of a single application or database. Use in additional applications requires separate licenses for the use of this Software.
- c. Distribution or dissemination of the software license serial is strictly prohibited. This license grants the original licensee sole use of the license serial for enabling this Software.

**3. Restrictions:** You may not distribute copies of the Software to others (except as an integral part of a database or application within the terms of this License) or electronically transfer the Software from one computer to another over a network. You may distribute copies of the Software as an integral part of a development shell or non-compiled commercial database as long as

the DSTi copyright notices and documentation remain intact with the distribution. The Software contains trade secrets and to protect them you may not decompile, reverse engineer, disassemble, or otherwise reduce the Software to a human perceivable form. You may not modify, adapt, translate, rent, lease, loan or resell for profit the software or any part thereof.

**4. Termination:** This license is effective until terminated. This license will terminate immediately without notice from DSTi if you fail to comply with any of its provisions. Upon termination you must destroy the Software and all copies thereof, and you may terminate this license at any time by doing so.

**5. Update Policy:** DSTi may create, from time to time, updated versions of the Software. At its option, DSTi will make such updates available to the Licensee.

**6. Warranty Disclaimer:** The software is provided "as is" without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. DSTi does not warrant, guarantee, or make any representations regarding the use, or the results of the use, of the software or written materials in the terms of correctness, accuracy, reliability, currentness or otherwise. The entire risk as to the results and performance of the software is assumed by the Licensee. If the software or written materials are defective you, and not DSTi or its dealers, distributors, agents, or employees, assume the entire cost of all necessary servicing, repair or correction. No oral or written information or advice given by DSTi, its dealers, distributors, agents, or employees shall create a warranty or in any way increase the scope of this warranty, and you may not rely on such information or advice. This warranty gives you specific legal rights. You may have other rights, which vary from state to state.

**7. Governing Law:** This agreement shall be governed by the laws of the State of Florida.

# Copyrights and Trademarks

All trade names referenced in this document are the trademark or registered trademark of their respective holder.

BASh and TCP Deux are copyright Deep Sky Technologies, Inc.

4th Dimension, ACI, ACI US, 4D Compiler, 4D, 4D Server, 4D Client, and 4D Insider are trademarks of 4D, Inc.

4D Internet Commands plugin provided courtesy, and with permission, of 4D, Inc.

Internet Commands plugin provided courtesy, and with permission, of Christian Quest.

Macintosh and MacOS are trademarks of Apple Computer, Inc.

Windows is a trademark of Microsoft Corporation.

# Table of Contents

Items in **Bold** were just added in the latest release of the TCP Deux component.  
Items in *Italic* have not had their content filled in completely as of yet.  
Items in Underline have had important updates since the last release of the TCP Deux component.

Software License and Limited Warranty

Copyrights and Trademarks

Table of Contents

Preface

*About this Manual*

Acknowledgements

*Features*

System Requirements

Support

Components

*Compatibility Matrix*

Installing and Updating TCP Deux

Managing Installation Conflicts

Affix TCP Deux Document

4D Internet Commands Plugin

Internet ToolKit Plugin

Uninstalling TCP Deux

TCP and TCP Deux Conventions

TCPd Streams Stack

HOSTs

IP Addresses

Constants

TCPd\_Plugin\_Types

TCPd\_Protocols

*Methods*

INIT\_TCPd

RES\_Open\_TCPd

TCPd\_Close\_Stream

TCPd\_Close\_Streams\_by\_Protocol

TCPd\_Copy\_s

TCPd\_Count\_Rows\_s

TCPd\_ERROR

TCPd\_Get\_DNSLookup

TCPd\_Get\_DNSLookup\_Reverse

TCPd\_Get\_Index\_s

TCPd\_Get\_LocalPort\_s

TCPd\_Get\_ProcessID\_s  
TCPd\_Get\_Protocol\_Count\_s  
TCPd\_Get\_Protocol\_s  
TCPd\_Get\_Status  
TCPd\_Get\_Status\_s  
TCPd\_Get\_StreamRef\_by\_Index\_s  
TCPd\_Get\_Stream\_Information  
TCPd\_Get\_TCP\_Info  
TCPd\_Open\_Listener  
TCPd\_Open\_Stream  
TCPd\_qi\_Handler\_Busy\_s  
TCPd\_qi\_INITed  
TCPd\_Receive\_File  
TCPd\_Receive\_to\_BLOB  
TCPd\_Send\_BLOB  
TCPd\_Send\_File  
TCPd\_Send\_Text  
TCPd\_Set\_ProcessID\_by\_Index\_s  
TCPd\_Set\_Status\_by\_Index\_s  
TCPd\_Update\_Statuses\_by\_Prot\_s  
TCPd\_Wait\_for\_NotStatus  
TCPd\_Wait\_for\_Status

#### Version History

TCP Deux v1.0.0b01

*Using TCP Deux*

*TCP Deux Error Codes*

*Index*

# Preface

The TCP Deux component is designed to work in conjunction with many other components. Specifically, the TCP Deux component requires that the BASH component, available for free from DSTi, be installed already in your database structure file.

Make certain that you view the compatibility matrix for components available to make certain you are using compatible versions of the different components required. There is a compatibility matrix available in this manual; the most recent compatibility matrix is available on the DSTi web site.

Other optional components which can be used with TCP Deux include SMTP Deux, POP3 Deux, and FTP Deux, among others. Details and documentation for these components are provided separately.



## About this Manual

asd

# Acknowledgements

The creation of the TCP Deux component is not directly attributable to any single person. Particular pieces of functionality within the TCP Deux component may be from the direct knowledge and experience of certain developers, but the overall concept and construction of the TCP Deux component has come from all of the developers at Deep Sky Technologies, Inc.

In particular, the tireless efforts of Robert T. McGoye have contributed the most to the TCP Deux component. His ability, and patience, to be able to tolerate the swings in the atmosphere at DSTi, have proven to be invaluable in the development of the TCP Deux component.

Later tweaks and additions to the TCP Deux component have resulted from the training of James T. Crate. Mr. Crate's experience in many different programming environments has provided refreshing insights into the overall structure and organization of the core routines at DSTi, the same core routines which are available in the BASH and TCP Deux components.

Finally, I, Steven G. Willis, might have had something to do with the creation of the TCP Deux component...



# Features

asd

# System Requirements

The TCP Deux component is compatible with both Macintosh and Windows installations of 4th Dimension.

Since it is a component, it does require at least version 6.7 of 4th Dimension or above, including 4D Insider v6.7 or above for installation.

Other than the normal hardware and software requirements for your version of 4th Dimension, there are no other minimum requirements for proper use of this software.

# Support

Support is provided for TCP Deux component free of charge for all currently licensed users. Included support services provided for all currently licensed users encompasses all of the online support services available through the DSTi web site (email, FAQ, messaging, etc.). Check the DSTi web site for current direct support options available; we are always working to offer more resources for your needs.

Contact information, including email address(es), phone number(s), and a Contact Us request form, for Deep Sky Technologies, Inc., can be found on the DSTi web site located at:

<http://www.deepskytech.com/>

If there are terms or conventions which you find difficult to understand in relation to the TCP Deux component or TCP networks in general, feel free to contact Deep Sky Technologies, Inc., support. We will be more than happy to help you in any way we reasonably can. And, only through your questions do we know what subjects to include in future versions of this manual.



# Components

A component groups various 4D objects (tables, project methods, forms, menu bars, variables, etc.) representing one or more additional functions. Developing a 4D component providing electronic mail functionality is one such example. A component is autonomous and must be able to be installed in any 4D structure.

Components are defined, generated, and installed with the help of 4D Insider. The component definition is based on the cross referencing analysis performed by 4D Insider (target objects and source objects).

Unlike libraries and groups, components embed the idea of security of objects that they compose. During the development phase of the component, each object is attributed an access type, "Public", "Protected" or "Private". This attribute determines whether each object will be visible or modifiable in 4th Dimension and in 4D Insider once the component is installed within a 4D database.



# Compatibility Matrix

asd

# Installing and Updating TCP Deux

Installing TCP Deux or updating an existing version of TCP Deux within a 4D database is performed using 4D Insider. The activity primarily consists of installing the TCP Deux component in a database structure opened with 4D Insider (installing the TCP Deux component in a library is not supported at this time).

4D Insider will manage possible conflict issues within the installation and will inform you as they are detected. Though, with the naming conventions used within the TCP Deux component and the limited number of object names, conflicts should be very rare.

To install or update the TCP Deux component, follow these very simple steps:

**Open the uncompiled structure that you wish to install TCP Deux into using 4D Insider.**

**Choose the "Install/Update..." command in the "Components" menu.**

A standard open file dialog box will appear.

**Select the TCP Deux component file and click on the "Open" button.**

4D Insider parses the TCP Deux component and prepares to integrate it with your open database. 4D Insider will detect if the operation is an installation or an update of the TCP Deux component.

In the event of a new installation, all TCP Deux objects are installed.

In the event of an update, 4D Insider compares the version numbers of both the currently installing TCP Deux component and the already installed TCP Deux component. If the date of the "new" component is older than the already installed component, a dialog box will

alert you, allowing you to then "Continue" or "Cancel" the update.

4D Insider replaces old objects with newer objects within the TCP Deux component and adds new objects from the new TCP Deux component. 4D Insider takes into account "public" objects having been modified by you (e.g. "\_ERROR" methods) and will prompt you to either save or replace them. If any other conflicts arise from the installation or update of the TCP Deux component, 4D Insider will prompt you with an appropriate dialog box.

**Save the database in 4D Insider.**

**Place a copy of the Affix TCP Deux document in the 4DX folder.**

The Affix TCP Deux document (entitled **Affix\_TCP\_Deux** on Macintosh) contains essential data, resources and constants for many of the methods within the TCP Deux component. For many of the methods within the TCP Deux component to function properly, the Affix TCP Deux document must be in the 4DX folder for the current structure.

On Macintosh, the Affix TCP Deux document is entitled **Affix\_TCP\_Deux** and is located in the Mac4DX directory in the TCP Deux component's archive. The document should be copied into the Mac4DX folder of your current structure. If the TCP Deux component is going to be used in all of your 4D projects, the Affix TCP Deux document can instead be placed within the Mac4DX folder within the 4D folder of your system.

On Windows, the Affix TCP Deux document is actually two documents: **Afx\_TCPd.4DX** and **Afx\_TCPd.RSR**. These two documents correspond to the data fork and the resource fork of the Affix TCP Deux document used on Macintosh. These documents are located in the Win4DX directory in the TCP Deux component's archive. These documents should be copied into the Win4DX folder of your current structure. If the TCP Deux component is going to be used in all of your 4D projects, the Affix TCP

Deux document can instead be placed within the Win4DX folder within the 4D folder of your system.

For client/server installations in cross-platform environments, both the Macintosh and Windows versions of the Affix TCP Deux document should be installed.

**Place copies of the appropriate TCP connectivity plugins in the 4DX folder.**

Since the TCP Deux component is compatible with both 4D Internet Commands and Internet ToolKit, copies of both of these plugins must be installed in your 4DX folder(s). But, at any given installation, only one plugin will be utilized. Plugin stubs have been provided for each plugin to reduce the total software package size which you require for any particular plugin combination use.

Please the sections of this manual dealing with 4D Internet Commands Plugin and Internet ToolKit Plugin, below, for more details on exactly which plugin documents should be installed for your particular installation needs.

**Call the method *INIT\_TCPd* early in the On Startup database method.**

To initialize the TCP Deux component in your code, place a call to the method *INIT\_TCPd* early in your **On Startup** database method.

Details about the *INIT\_TCPd* method can be found in the method documentation section of this manual, below.

The TCP Deux component is now installed/updated in your database and is listed on the "Components" page of the 4D Explorer.

## **Managing Installation Conflicts**

On very rare occasions, when the TCP Deux component is installed or updated in your 4D database, several questions and conflicts may arise. In the event of an update, 4D Insider will detect that you have modified one or more "Public" objects in TCP Deux after the initial installation. Or, one or more objects of the same type and of the same name may already exist in your database and in the TCP Deux component.

4D Insider detects and solves these conflicts during installation:

### **Modified public objects (updates only)**

In this case, 4D Insider alerts you by a dialog box, allowing you to choose an update mode:

Replace the object

Replace all objects

Do not replace the object

Stop installation

### **Name conflicts**

In this case, 4D Insider stops the TCP Deux installation process, alerts you through a dialog box and saves the list of objects in conflict. This list is stored as a text file in the 4D database folder.

Naming conflicts between logical objects, such as variables, are managed by 4D Insider, in a manner that allows database compilation and avoids conflicts between TCP Deux and other 4D components.

It may be necessary to rename certain objects in your database or in other components in order to be able to install the TCP Deux .

If any naming conflicts do occur between TCP Deux and other 4D components, please notify Deep Sky Technologies, Inc., immediately.

## **Affix TCP Deux Document**

The Affix TCP Deux document (entitled **Affix\_TCP\_Deux** on Macintosh) contains essential data and resources for many of the methods within the TCP Deux component. For many of the methods within the TCP Deux component to function properly, the Affix TCP Deux document must be in the 4DX folder for the current structure. For distributed and installed versions of your 4D projects, the Affix TCP Deux document must be available in the 4DX folder for the TCP Deux methods to continue to function properly.

**Note:** it is best to consider the Affix TCP Deux document another plug-in within your 4D project. Though there no actual plug-in calls available within the Affix TCP Deux document, it does contain data and resources essential to the operation of the TCP Deux methods. The TCP Deux component has been designed to find the Affix TCP Deux document correctly in all environments (any platform, any 4DX folder, single user or clients/server, etc.). Since the Affix TCP Deux document is configured similarly to plug-ins, 4D and the TCP Deux component will automatically manage the document for you in all of the possible installations of a 4D project.

## **4D Internet Commands Plugin**

The 4D Internet Commands plugin is a plugin available from 4D, Inc., and 4D SA for providing network connectivity within 4D based applications. TCP Deux includes two copies of this plugin for each platform.

The first copy of the plugin (entitled **ic\_v670** and **ic\_v670.4dx**, on Macintosh and Windows, respectively) is for use only if your application *will* be using 4D Internet Commands to provide network connectivity.

The second copy of the plugin (entitled **ic\_v670\_stub** and **ic\_v670\_stub.4dx**, on Macintosh and Windows, respectively) is for use only if your application *will not* be using 4D Internet Commands to provide network connectivity. It is a placeholder for the compilation in the event that your application will be using either Internet ToolKit v2.0.x or Internet ToolKit v2.5.x.

The 4D Internet Commands plugin is provided unmodified directly from 4D, Inc., and 4D SA. The 4D Internet Commands plugin stub document is made available by DSTi for compilation compatibility.

## Internet ToolKit Plugin

The TCP Deux component is compatible with both Internet ToolKit v2.0.x and Internet ToolKit v2.5.x. There are significant differences between these two version of the ITK plugin. But, the copies of the these plugins provided with TCP Deux make almost all of these differences transparent (the only exception being support for SSL connections).

Depending on which version of ITK your application may support, different plugin documents are provided.

NOTE: if your application will be using 4D Internet Commands for network connectivity, refer to the section for ITK v2.5.x for the correct ITK stub to install.

### ITK v2.0.x

If your application will use ITK v2.0.x for network connectivity, install the ITK v2.0.x version of the plugin contained within the TCP Deux software package. This version is entitled **itkp\_v204** on Macintosh and **itkp\_v204.4dx** on Windows.

In this event, no copy of ITK v2.5.x need be installed with your application for TCP Deux to function properly. But, make certain that your application uses the version of the ITK v2.0.x plugin which is included with the TCP Deux component. Specific modifications

have been made to the ITK v2.0.x plugin within these copies of the plugin to provide compiler compatibility within the TCP Deux package.

NOTE: if your application will be using ITK v2.0.x for network connectivity, refer to the section for the 4D Internet Commands Plugin to install the 4D Internet Commands plugin stub, also.

## ITK v2.5.x

If your application will use ITK v2.5.x for network connectivity, install the ITK v2.5.x version of the plugin contained within the TCP Deux software package. This version is entitled **itk\_v250b10** on Macintosh and **itk\_v250b10.4dx** on Windows. Keep in mind that this version of the ITK plugin is the latest beta and is set to expire on March 31st, 2001.

If your application will use 4D Internet Commands for network connectivity, installation of the ITK plugin stub is required. For simplicity, the ITK plugin stub is provided in ITK v2.5.x format only. The ITK v2.5.x plugin stub is entitled **itk\_v250b10\_stub** on Macintosh and **itk\_v250b10\_stub.4dx** on Windows.

NOTE: if your application will be using ITK v2.5.x for network connectivity, refer to the section for the 4D Internet Commands Plugin to install the 4D Internet Commands plugin stub, also.



# Uninstalling TCP Deux

4D Insider allows you to uninstall the TCP Deux component from your 4D database.

To uninstall TCP Deux from your 4D database:

Using 4D Insider, open your database containing the copy of TCP Deux to be uninstalled.

In the "Main" listing window, select the TCP Deux component.

Consider again how great the TCP Deux component is and make certain that you will *really* no longer need it in your 4D database.

Select the "Uninstall..." command in the "Components" menu.

This command is only active when a component is installed in the database. A dialog box appears allowing you to confirm or cancel the operation. If you uncertain about the previous step then the cancel option is probably your best choice at this time.

Click "OK" to validate the operation.

Remove the Affix TCP Deux document and internet connectivity plugins from your 4DX folder.

Remove the call to the method *INIT\_TCPd* from your On Startup database method.

All objects from the TCP Deux component are deleted from your 4D database. Obviously, you are now very sad to no longer have the TCP Deux component in your 4D database. Crying is allowed...



## TCP and TCP Deux Conventions

Throughout this manual, and all other documentation and supporting materials, included with the TCP Deux component package, there are different core knowledge which is essential to know and understand. With this knowledge, basically concerning the conventions used on TCP networks and conventions used within the TCP Deux component, you will be able to more easily and efficiently utilize the functionality available within this software package.

If there are other terms or conventions which you find difficult to understand in relation to the TCP Deux component or TCP networks in general, feel free to contact Deep Sky Technologies, Inc., support. We will be more than happy to help you in any way we reasonably can. And, only through your questions do we know what subjects to include in future versions of this manual.

# TCPd Streams Stack

The TCPd Streams Stack is an internal mechanism within the TCP Deux component to keep efficiently keep track of open TCP streams within 4th Dimension. Whenever the routines within the TCP Deux component are used to open, change, or close a TCP stream, the TCPd Streams Stack is updated to reflect directly the last known state of all applicable TCP streams managed by TCP Deux.

The TCPd Streams Stack is accessible within your 4th Dimension code by using the accessor routines provided within the TCP Deux component. All of the methods which work directly with the TCPd Streams Stack can be identified by the trailing “\_s” within the name of the routine (e.g. TCPd\_Copy\_StreamStack\_s, TCPd\_Get\_ProcessID\_s, etc.).

Many other methods within the TCP Deux component may access or make use of the TCPd Streams Stack, though that is not the primary function of such methods. Rather, these other methods within the TCP Deux component work to maintain the integrity and accuracy of the TCPd Streams Stack.

The TCPd Streams Stack is basically just a listing of data about each TCP stream. For each TCP stream managed by the TCP Deux component, there is one row in the TCPd Streams Stack. Each row in the TCPd Streams Stack contains a single field for each of the following pieces of information:

<u>Field Name</u>	<u>Type</u>
Stream Reference	Longint
Port (Local)	Longint
Protocol	Longint
Status	Longint
IP Address (Local)	Longint
Handler Process ID	Longint

The stream reference is the unique identified used to reference each TCP communications stream available.

The port is the local port which is used for the TCP communications.

The protocol is the coded protocol value which is to be used on the TCP communications stream. See the section *TCPd\_Protocols* in

this manual for details about the different coded values available for the protocols field.

Status is the TCP stream status. The following is a listing of all of the possible stream status values:

<u>Status</u>	<u>Name</u>	<u>Description</u>
0	Closed	no connection
2	Listen	listening for incoming connection
4	SYN received	incoming connection being established
6	SYN sent	outgoing connection being established
8	Established	connection up
10	FIN Wait 1	connection up; close sent
12	FIN Wait 2	connection up; close sent and acknowledged
14	Close Wait	connection up; close received
16	Closing	connection up; close issued and received
18	Last ACK	connection up; close issued and received
20	Time Wait	connection being broken

IP address is the local IP address of the TCP communications stream. A value of zero (0) within this field indicates that every IP local IP address on the machine is available for use for the TCP communications stream, though outgoing connections will use the primary IP address of the machine in most cases.

Handler process ID is the 4D process which is set to handle the TCP communications process.

**Note:** the handler process ID field is provided merely for informational, and convenience, purposes. There is direct affect this field's value has on the TCP communications handling.

# HOSTs

Host names on a TCP network can refer to a few different formatted string values.

One format for a valid host name is the dotted IP address of the network device, e.g. "63.175.177.37". Another valid host name format is the domain name of the network device, e.g. "deepskytech.com".

Both host name formats are valid host names to be used for addressing a device on a TCP network. Any parameters in the TCP Deux component which require a host name refer to a host name that fits into either of these formats.

## IP Addresses

Throughout the TCP Deux component package, IP addresses for local and remote hosts are handled in 4D as longint values. This provides a much more convenient and memory efficient means for managing IP addresses throughout the TCP Deux component package. When an IP address is required for a specific parameter in a TCP Deux routine, it is assumed that the longint encoding of the IP address is the valid value format.

The BASH component package contains routines to quickly and easily convert between IP addresses stored as longint values and IP addresses stored as string values (dotted IP addresses). The routines *CONV\_IP\_to\_Longint* and *CONV\_Longint\_to\_IP* allow for the conversion of a single value of one type to another.

The following is a listing of sample IP addresses and their corresponding longint values in 4th Dimension:

<u>Dotted IP Address</u>	<u>Longint IP Address</u>
0.0.0.0	0
0.0.0.1	1
0.0.0.2	2
0.0.1.0	256
0.0.1.1	257
1.1.1.1	16843009
127.255.255.255	2147483647
128.0.0.0	-2147483648

128.0.0.1	-2147483647
255.255.255.255	-1
63.175.177.37	1068478757





# Constants

There are a minimal number of custom constants included with the TCP Deux component package. These constants are grouped into a few convenient constant groups for easier referencing and organization.

Where appropriate, it is highly recommended that the custom constants included with the TCP Deux component be utilized within your code; this will simplify considerably future feature enhancements to the core code within TCP Deux.

## TCPd\_Plugin\_Types

The TCPd\_Plugin\_Types constants group contains one constant for each of the supported TCP plugins supported within the TCP Deux component package. The following is a listing of the constants, and their values, within the TCPd\_Plugin\_Types constant group:

<u>Constant</u>	<u>Value</u>
TCPd_IC_v67x_Plugin	1
TCPd_ITK_v25x_Plugin	2
TCPd_ITK_v20x_Plugin	4

These constants are used only when initializing the TCP Deux component. The first parameter to the TCP Deux method *INIT\_TCPd* takes one of these constant values to indicate which TCP plugin is to be used in the current 4D application for all TCP communications.

## TCPd\_Protocols

The TCPd\_Protocols constants group contains different constants for each commonly used TCP protocol utilized. There are distinct constants for opening a TCP stream for sending or receiving (client or server, remote or host, session or listen, etc.).

The protocol to be used for a specific TCP stream is important to set correctly. To properly support SSL communications, for instance, when using Internet Toolkit v2.5.x, setting an improper protocol for a TCP stream may prevent the SSL encoding to work properly.

The following is a listing the constants, and their values, within the TCPd\_Protocols constant group:

<u>Constant</u>	<u>Value</u>
TCPd_HTTP_Listen	1
TCPd_HTTP_Session	2
TCPd_SMTP_Listen	3
TCPd_SMTP_Session	4
TCPd_POP3_Listen	5
TCPd_POP3_Session	6
TCPd_FTP_Listen	7
TCPd_FTP_Session	8
TCPd_HTTPS_Listen	9
TCPd_HTTPS_Session	10
TCPd_other_Listen	601
TCPd_other_Session	602

If the specific protocol which is to be used on a TCP communications stream is not listed in TCPd\_Protocols constants group already, the "other" constants are available.



# Methods

asd



## INIT\_TCPd

**INIT\_TCPd** (*Plugin to Use* ; *Macintosh Plugin Serial* ; *Windows Plugin Serial* ; *TCP Deux Serial* )

**INIT\_TCPd**  
(  
    -> *Plugin to Use* : **Longint**  
    -> *Macintosh Plugin Serial* : **Text**  
    -> *Windows Plugin Serial* : **Text**  
    -> *TCP Deux Serial* : **Text**  
)

Parameter	Type	Description
<i>Plugin to Use</i>	Longint	TCP plugin to use with TCP Deux (values are available in the TCP Deux constants group <i>TCPd_Plugin_Types</i> )
<i>Macintosh Plugin Serial</i>	Text	Macintosh or primary ITK v2.5.x plugin serial
<i>Windows Plugin Serial</i>	Text	Windows or SSL ITK v2.5.x plugin serial
<i>TCP Deux Serial</i>	Text	TCP Deux serial

The method **INIT\_TCPd** initialises the TCP Deux component. A single call to this method should be made early in the On Startup database method in your 4D application. Make certain the call to this method follows the initialization call to the BASH component but before any other calls to the TCP Deux component package.

Any initialization and serialization of the TCP plugin being used will be handled directly by this method. So, any existing calls which exist to initialise and/or serialise, for instance, ITK can be removed from your 4D application.

*Plugin to Use* is the coded TCP plugin which is to be used with the TCP Deux component for all TCP communications. See the section *TCPd\_Plugin\_Types* , in this manual, for details about the different plugin code constants available for use with this parameter.

*Macintosh Plugin Serial* is the first plugin serial for the TCP plugin being used. For ITK v2.0.x, the Macintosh specific serial should be passed in this parameter. For ITK v2.5.x, the primary ITK serial should be passed in this parameter. For IC, this parameter can be left empty.

*Windows Plugin Serial* is the second plugin serial for the TCP plugin being used. For ITK v2.0.x, the Windows specific serial should be passed in this parameter. For ITK v2.5.x, the SSL ITK serial should be passed in this parameter. For IC, this parameter can be left empty.

*TCP Deux Serial* parameter is the TCP Deux serial which came with your purchase of the TCP Deux component package. A single TCP Deux serial provides for use of the TCP Deux component package on all platforms. If the TCP Deux package is being tested or being used in demonstration mode, use the TCP Deux demo serial number provided from DSTi for use in this parameter.



## **RES\_Open\_TCPd**

**RES\_Open\_TCPd** => *Resource Fork File Reference*

**RES\_Open\_TCPd**

=> *Resource Fork File Reference: Time*

Parameter	Type	Description
<i>Resource Fork File Reference</i>	<b>Time</b>	<b>File reference to resource fork of TCP Deux Affix document</b>

The method **RES\_Open\_TCPd** returns the 4D resource document reference for the resource fork of the TCP Deux Affix document. This method will find the Affix document properly on any platform if it is stored in the

4DX folder next to the 4D structure or if it is stored in the 4D folder in the system.

This method is provided merely for use by the protocol component packages that are compatible with the TCP Deux component (e.g. SMTP Deux, POP3 Deux, FTP Deux, etc.).

*Resource Fork File Reference* is the 4D compatible resource fork document reference of the TCP Deux Affix document.

---

## **TCPd\_Close\_Stream**

**TCPd\_Close\_Stream** ( *Stream Reference* )

**TCPd\_Close\_Stream**

(  
    -> *Stream Reference* : Longint  
)

Parameter	Type	Description
<i>Stream Reference</i>	Longint	TCP stream reference to close

The method **TCPd\_Close\_Stream** will close an existing TCP communications stream. No regard will be given to the current status the TCP communication stream; the closing of the TCP communications stream is initiated immediately upon calling this method.

The entry in the TCPd Streams Stack for the TCP communications stream being closed is removed once the stream has completed closing.

*Stream Reference* is the TCP communications stream reference to be closed.

---

## **TCPd\_Close\_Streams\_by\_Protocol**

## ***TCPd\_Close\_Streams\_by\_Protocol*** ( *Protocol* )

***TCPd\_Close\_Streams\_by\_Protocol***  
(  
    -> *Protocol* : Longint  
)

Parameter	Type	Description
<i>Protocol</i>	Longint	Protocol of match for TCP streams to close

The method ***TCPd\_Close\_Streams\_by\_Protocol*** will close all TCP communications streams of a specified protocol. The protocol value is checked within the TCPd Streams Stack for every open TCP communications stream and existing streams using the specified protocol are then closed.

Entries in the TCPd Streams Stack for the TCP communications streams being closed are removed once each stream has completed closing.

This method is ideal for use when quitting the 4D application.

*Protocol* is the protocol to check within the TCPd Streams Stack to indicate which TCP communications streams are to be closed by this method. Valid values for the *Protocol* parameter include all of the protocol values in the constants group *TCPd\_Protocols* included with the TCP Deux component (documented separately in this manual).



## **TCPd\_Copy\_s**

***TCPd\_Copy\_s*** ( *Referenced Stream References* ; *Referenced Ports* ;  
*Referenced Protocols* ; *Referenced Statuses* ; *Referenced IP*  
*Addresses* ; *Referenced Handler Process IDs* )

***TCPd\_Copy\_s***  
(  
    -> *Referenced Stream References* : Pointer



- > *Referenced Ports* : **Pointer**
- > *Referenced Protocols* : **Pointer**
- > *Referenced Statuses* : **Pointer**
- > *Referenced IP Addresses* : **Pointer**
- > *Referenced Handler Process IDs* : **Pointer**

)

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Referenced Stream References</i>	<b>Pointer</b>	Pointer to longint array to hold TCP stream references
<i>Referenced Ports</i>	<b>Pointer</b>	Pointer to longint array to hold ports
<i>Referenced Protocols</i>	<b>Pointer</b>	Pointer to longint array to hold protocols
<i>Referenced Statuses</i>	<b>Pointer</b>	Pointer to longint array to hold TCP stream statuses
<i>Referenced IP Addresses</i>	<b>Pointer</b>	Pointer to longint array to hold remote IP addresses
<i>Referenced Handler Process IDs</i>	<b>Pointer</b>	Pointer to longint array to hold handler process IDs

The method *TCPd\_Copy\_s* provides a means for a separate copy of the current state of the TCPd Streams Stack to be made. It is ideal for providing a TCP streams monitor interface which is updated regularly for programming and debugging purposes.

*Referenced Stream References* should be a pointer to a longint array to hold the list of stream references in the TCPd Streams Stack. If this value is a NULL pointer, there will be no listing of the stream references within the TCPd Streams Stack provided with this method call.

*Referenced Ports* should be a pointer to a longint array to hold the list of local ports in the TCPd Streams Stack. If this value is a NULL pointer, there will be no listing of the local ports within the TCPd Streams Stack provided with this method call.

*Referenced Protocols* should be a pointer to a longint array to hold the list of protocols in the TCPd Streams Stack. If this value is a NULL pointer, there will be no

listing of the protocols within the TCPd Streams Stack provided with this method call.

*Referenced Statuses* should be a pointer to a longint array to hold the list of TCP communications stream statuses in the TCPd Streams Stack. If this value is a NULL pointer, there will be no listing of the TCP communications stream statuses within the TCPd Streams Stack provided with this method call.

*Referenced IP Addresses* should be a pointer to a longint array to hold the list of local IP addresses in the TCPd Streams Stack. If this value is a NULL pointer, there will be no listing of the local IP addresses within the TCPd Streams Stack provided with this method call.

*Referenced Handler Process IDs* should be a pointer to a longint array to hold the list of 4D handler process IDs in the TCPd Streams Stack. If this value is a NULL pointer, there will be no listing of the 4D handler process IDs within the TCPd Streams Stack provided with this method call.



## **TCPd\_Count\_Rows\_s**

**TCPd\_Count\_Rows\_s** => *Stream Stack Row Count*

**TCPd\_Count\_Rows\_s**

=> *Stream Stack Row Count: Longint*

Parameter	Type	Description
<i>Streams Row Count</i>	Longint	Number of rows in TCPd Streams Stack

The method **TCPd\_Count\_Rows\_s** returns the number of rows currently in the TCPd Streams Stack.

This method is ideal for use when quitting the 4D application.

*Stream Stack Row Count* is the number of rows currently in the TCPd Streams Stack.



## **TCPd\_ERROR**

**TCPd\_ERROR** (*TCPd Error Number* ; *Special Error Text* ; *Calling Method Name* )

**TCPd\_ERROR**  
(  
    -> *TCPd Error Number*: **Longint**  
    -> *Special Error Text*: **Text**  
    -> *Calling Method Name*: **Text**  
)

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>TCPd Error Number</i>	<b>Longint</b>	Internal TCPd error number
<i>Special Error Text</i>	<b>Text</b>	Special text to describe the exact error instance
<i>Calling Method Name</i>	<b>Text</b>	Name of the method that the error condition occurred in

The method **TCPd\_ERROR** acts as a callback method from within the TCP Deux component for errors that may occur. Any time an error condition is detected within the TCP Deux, a call to the method **TCPd\_ERROR** is made.

The internal *TCPd Error Number* is passed to this method as the first parameter. The *Special Error Text* parameter will contain any relevant error text which is specific to the error which occurred. It is not uncommon for the *Special Error Text* value to be empty. The *Calling Method Name* will always contain the name of the TCP Deux method which called the **TCPd\_ERROR** method.

The **TCPd\_ERROR** method has been implemented as a source for a consistent interface and/or error tracking mechanism to be available while using the TCP Deux component. This method can be modified to suit the needs of the database in which the TCP Deux component has been installed.



## TCPd\_Get\_DNSLookup

**TCPd\_Get\_DNSLookup** ( *Domain Name* ; *Mode* ) => *IP Address*

**TCPd\_Get\_DNSLookup**  
(  
    -> *Domain Name* : **Text**  
    -> *Mode* : **Longint**  
)  
    => *IP Address*: **Longint**

Parameter	Type	Description
<i>Domain Name</i>	<b>Text</b>	Domain name to lookup
<i>Mode</i>	<b>Longint</b>	Mode indicator for which IP address to return (ITK only)
<i>IP Address</i>	<b>Longint</b>	IP address returned

The method **TCPd\_Get\_DNSLookup** provides DNS lookup capabilities to convert a domain name to a valid IP address. The conversion is done by checking the DNS listing of the specified domain to retrieve the IP address of the TCP device serving for the specified domain name.

*Domain Name* is the valid domain name to be lookup up.

*Mode* (supported with ITK only) is the coded parameter value to indicate what action to take for load balanced (LB) DNS entries for a particular domain name. The following table lists the valid values for the *Mode* parameter and the functionality associated with each value:

<u>Mode</u>	<u>Action</u>
- 1	returns random LB IP address
0 or 1	returns first LB IP address
2	returns second LB IP address
3	returns third LB IP address
4	returns fourth LB IP address

*IP Address* is the IP address retrieved from the DNS for the specified *Domain Name* .



## TCPd\_Get\_DNSLookup\_Reverse

**TCPd\_Get\_DNSLookup\_Reverse** ( *IP Address* ; *Mode* ) => *Domain Name*

**TCPd\_Get\_DNSLookup\_Reverse**  
(  
    -> *IP Address* : Longint  
    -> *Mode* : Longint  
)  
=> *Domain Name*: Text

Parameter	Type	Description
<i>IP Address</i>	Longint	IP address to lookup
<i>Mode</i>	Longint	Mode indicator for domain name format to return
<i>Domain Name</i>	Text	Domain name returned

The method **TCPd\_Get\_DNSLookup\_Reverse** provides reverse DNS lookup capabilities to convert an IP address to a valid domain name. The conversion is done by checking the DNS listing of the specified IP address to retrieve the reverse DNS record for the specified IP address.

Refer to the manual accompanying your DNS server for instructions for enter proper reverse DNS zones and records.

*IP Address* is the IP address to do a reverse DNS lookup upon.

*Mode* is the reverse lookup options to use when returning values for the reverse DNS lookup. The following table lists the valid values for the *Mode* parameter and the functionality associated with each value:

Mode	Action
0	returns full domain name; if none exist, dotted IP address is returned
1	returns full domain name; if none exist, NULL is returned
2	returns dotted IP address; NOTE: no TCP activity needed with this mode

*Domain Name* is the reverse DNS full domain name, or dotted IP address depending on the options specified, for the IP address specified.



## **TCPd\_Get\_Index\_s**

**TCPd\_Get\_Index\_by\_s** (*Stream Reference* ) => *Stream Stack Index*

**TCPd\_Get\_Index\_by\_s**

(  
    -> *Stream Reference* : Longint  
)  
    => *Stream Stack Index*: Longint

Parameter	Type	Description
<i>Stream Reference</i>	Longint	Stream reference to lookup in TCPd Streams Stack
<i>Stream Stack Index</i>	Longint	Index into TCPd Streams Stack for specified stream reference

The method **TCPd\_Get\_Index\_by\_s** returns the index in the TCPd Streams Stack matching the specified stream reference provided.

*Stream Reference* is the TCP communication stream reference to look for in the TCPd Streams Stack.

*Stream Stack Index* is the positive longint index value in the TCPd Streams Stack for the row matching the provided TCP communications stream reference *Stream Reference* . If the value for *Stream Reference* is not found in the TCPd Streams Stack, *Streams Stack Index* will be set to negative one (-1).



## **TCPd\_Get\_LocalPort\_s**

**TCPd\_Get\_Local\_Port\_f\_Stack\_s** (*Stream Reference* ) => *Port*

**TCPd\_Get\_Local\_Port\_f\_Stack\_s**

```
(
    -> Stream Reference : Longint
)
=> Port: Longint
```

Parameter	Type	Description
<i>Stream Reference</i>	Longint	Stream reference to lookup in TCPd Streams Stack
<i>Port</i>	Longint	Port listed in TCPd Streams Stack for specified stream reference

The method *TCPd\_Get\_LocalPort\_s* returns the local port in the TCPd Streams Stack matching the specified stream reference provided.

*Stream Reference* is the TCP communication stream reference to look for in the TCPd Streams Stack.

*Port* is the local port value in the TCPd Streams Stack for the row matching the provided TCP communications stream reference *Stream Reference* . If the value for *Stream Reference* is not found in the TCPd Streams Stack, *Port* will be set to negative one (-1).



## TCPd\_Get\_ProcessID\_s

*TCPd\_Get\_ProcessID\_s* ( *Stream Reference* ) => *Handler Process ID*

***TCPd\_Get\_ProcessID\_s***

```
(
    -> Stream Reference : Longint
)
=> Handler Process ID: Longint
```

Parameter	Type	Description
<i>Stream Reference</i>	Longint	Stream reference to lookup in TCPd Streams Stack
<i>Handler Process ID</i>	Longint	Handler process ID listed in TCPd Streams Stack for specified stream reference

The method *TCPd\_Get\_ProcessID\_s* returns the 4D handler process ID in the TCPd Streams Stack matching the specified stream reference provided.

*Stream Reference* is the TCP communication stream reference to look for in the TCPd Streams Stack.

*Handler Process ID* is the 4D handler process ID in the TCPd Streams Stack for the row matching the provided TCP communications stream reference *Stream Reference*. If the value for *Stream Reference* is not found in the TCPd Streams Stack, *Handler Process ID* will be set to negative one (-1).

---

## TCPd\_Get\_Protocol\_Count\_s

*TCPd\_Get\_Protocol\_Count\_s* ( *Protocol* ) => *Streams Count*

```
TCPd_Get_Protocol_Count_s
(
    -> Protocol : Longint
)
=> Streams Count: Longint
```

Parameter	Type	Description
<i>Protocol</i>	Longint	Protocol to lookup in TCPd Streams Stack
<i>Streams Count</i>	Longint	Number of entries in TCPd Streams Stack matching specified protocol

The method *TCPd\_Get\_Protocol\_Count\_s* returns the number of TCP communication streams currently in the TCPd Streams Stack using the specified protocol.

*Protocol* is the protocol to check within the TCPd Streams Stack to indicate which TCP communications streams are to be counted by this method. Valid values for the *Protocol* parameter include all of the protocol values in the constants group *TCPd\_Protocols* included with the TCP Deux component (documented separately in this manual).



*Streams Count* is the number of rows in the TCPd Streams Stack which are using the specified protocol *Protocol* .



## TCPd\_Get\_Protocol\_s

**TCPd\_Get\_Protocol\_s** (*Stream Reference* ) => *Protocol*

**TCPd\_Get\_Protocol\_s**

(  
    -> *Stream Reference* : Longint  
)  
    => *Protocol*: Longint

Parameter	Type	Description
<i>Stream Reference</i>	Longint	Stream reference to lookup in TCPd Streams Stack
<i>Protocol</i>	Longint	Protocol listed in TCPd Streams Stack for specified stream reference

The method *TCPd\_Get\_Protocol\_s* returns the protocol in the TCPd Streams Stack matching the specified stream reference provided.

*Stream Reference* is the TCP communication stream reference to look for in the TCPd Streams Stack.

*Protocol* is the protocol value in the TCPd Streams Stack for the row matching the provided TCP communications stream reference *Stream Reference* . If the value for *Stream Reference* is not found in the TCPd Streams Stack, *Protocol* will be set to negative one (-1). Valid values for the *Protocol* parameter include all of the protocol values in the constants group *TCPd\_Protocols* included with the TCP Deux component (documented separately in this manual).



## TCPd\_Get\_Status

**TCPd\_Get\_Status** ( *Stream Reference* ; *Selector Code* ) => *Stream Status*

**TCPd\_Get\_Status**

```
(  
    -> Stream Reference : Longint  
    -> Selector Code : Longint  
)  
=> Stream Status: Longint
```

Parameter	Type	Description
<i>Stream Reference</i>	Longint	Stream reference to check status of directly from TCP plugin
<i>Selector Code</i>	Longint	ITK status selector value (ITK only, see ITK manual for ITK_TCPStatus)
<i>Stream Status</i>	Longint	Status value for specified stream reference returned directly from TCP plugin

The method **TCPd\_Get\_Status** returns the TCP communications stream status as returned directly from the TCP plugin being used.

The current stream status of the specified TCP communications stream will be updated automatically within the TCPd Streams Stack within this method.

*Stream Reference* is the TCP communication stream reference to to check the status of with the current TCP plugin being used.

*Selector Code* (supported with ITK only) is the ITK selector code for retrieving the TCP communications stream status of. More details about *Selector Code* are available in the ITK plugin manual for the plugin method ITK\_TCPStatus. When using the IC plugin, this value is ignored.

*Stream Status* is the status of the TCP communications stream as obtained directly from the current TCP plugin. The following table lists the valid values and their meanings for the *Stream Status* parameter:

<u>Status</u>	<u>Name</u>	<u>Description</u>
---------------	-------------	--------------------

- 1	error	invalid stream reference
0	Closed	no connection
2	Listen	listening for incoming connection
4	SYN received	incoming connection being established
6	SYN sent	outgoing connection being established
8	Established	connection up
10	FIN Wait 1	connection up; close sent
12	FIN Wait 2	connection up; close sent and acknowledged
14	Close Wait	connection up; close received
16	Closing	connection up; close issued and received
18	Last ACK	connection up; close issued and received
20	Time Wait	connection being broken



## TCPd\_Get\_Status\_s

**TCPd\_Get\_Status\_s** ( *Stream Reference* ) => *Stream Status*

**TCPd\_Get\_Status\_s**

```
(
    -> Stream Reference : Longint
)
=> Stream Status: Longint
```

Parameter	Type	Description
<i>Stream Reference</i>	Longint	Stream reference to check status of directly from TCPd Streams Stack
<i>Stream Status</i>	Longint	Status value for specified stream reference returned directly from TCPd Streams Stack

The method **TCP\_Get\_Status\_s** returns the stream status in the TCPd Streams Stack matching the specified stream reference provided.

Using this method, no calls are made to the TCP plugin to update the stream status within the TCPd Streams Stack or to obtain the actual, direct stream status value from the current TCP plugin.

*Stream Reference* is the TCP communication stream reference to look for in the TCPd Streams Stack.

*Stream Status* is the stream status value in the TCPd Streams Stack for the row matching the provided TCP communications stream reference *Stream Reference* . If the value for *Stream Reference* is not found in the TCPd Streams Stack, *Stream Status* will be set to negative one (-1). The following table lists the valid values and their meanings for the *Stream Status* parameter:

<u>Status</u>	<u>Name</u>	<u>Description</u>
- 1	error	invalid stream reference
0	Closed	no connection
2	Listen	listening for incoming connection
4	SYN received	incoming connection being established
6	SYN sent	outgoing connection being established
8	Established	connection up
10	FIN Wait 1	connection up; close sent
12	FIN Wait 2	connection up; close sent and acknowledged
14	Close Wait	connection up; close received
16	Closing	connection up; close issued and received
18	Last ACK	connection up; close issued and received
20	Time Wait	connection being broken



## **TCPd\_Get\_StreamRef\_by\_Index\_s**

**TCPd\_Get\_StreamRef\_by\_Index\_s** ( *Stream Stack Index* ) => *Stream Reference*

**TCPd\_Get\_StreamRef\_by\_Index\_s**  
 (  
     -> *Stream Stack Index* : Longint  
 )  
     => *Stream Reference* : Longint

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Stream Stack Index</i>	Longint	Index into TCPd Streams Stack
<i>Stream Reference</i>	Longint	Stream reference within specified index of TCPd Streams Stack

The method **TCPd\_Get\_StreamRef\_by\_Index\_s** returns the stream reference in the TCPd Streams Stack matching the specified TCPd Streams Stack index provided.

*Stream Stack Index* is the row index to look for in the TCPd Streams Stack.

*Stream Reference* is the stream reference stored in the specified row *Stream Stack Index* within the TCPd Streams Stack. If the value of *Stream Stack Index* is invalid, out of range, or not found in the TCPd Streams Stack, then *Stream Reference* will be set to negative one (-1).



## **TCPd\_Get\_Stream\_Information**

**TCPd\_Get\_Stream\_Information** ( *Stream Reference* ; *Referenced Remote IP Address* ; *Referenced Remote Port* ; *Referenced Local Port* ; *Referenced SRTT* ; *Referenced Local IP Address* ) => *Error Code*

### **TCPd\_Get\_Stream\_Information**

(  
-> *Stream Reference* : **Longint**  
-> *Referenced Remote IP Address* : **Pointer**  
-> *Referenced Remote Port* : **Pointer**  
-> *Referenced Local Port* : **Pointer**  
-> *Referenced SRTT* : **Pointer**  
-> *Referenced Local IP Address* : **Pointer**  
)  
=> *Error Code*: **Longint**

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Stream Reference</i>	<b>Longint</b>	Stream reference to retrieve information about
<i>Referenced Remote IP Address</i>	<b>Pointer</b>	Referenced longint variable to hold remote IP address (ITK only)
<i>Referenced Remote Port</i>	<b>Pointer</b>	Referenced longint variable to hold remote port (ITK only)
<i>Referenced Local Port</i>	<b>Pointer</b>	Referenced longint variable to hold local port
<i>Referenced SRTT</i>	<b>Pointer</b>	Referenced longint variable to hold SRTT value (ITK only)

<i>Referenced Local IP Address</i>	<b>Pointer</b>	<b>Referenced longint variable to hold local IP address</b>
<i>Error Code</i>	<b>Longint</b>	<b>Error code returned from call</b>

The method *TCPd\_Get\_Stream\_Information* will return information pertaining directly to a specified stream reference. The data pertaining to the specified stream reference is retrieved directly from the current TCP plugin.

*Stream Reference* is the TCP communications stream reference to retrieve information about.

*Referenced Remote IP Address* (support by ITK only) is a pointer to a longint variable to contain the IP address of the remote host currently communicating on the specified TCP communications stream *Stream Reference*. If a NULL pointer is passed in the *Referenced Remote IP Address* parameter, then the remote IP address will not be retrieved or returned by this method.

*Referenced Remote Port* (support by ITK only) is a pointer to a longint variable to contain the port of the remote host currently communicating on the specified TCP communications stream *Stream Reference*. If a NULL pointer is passed in the *Referenced Remote Port* parameter, then the remote port will not be retrieved or returned by this method.

*Referenced Local Port* is a pointer to a longint variable to contain the port of the local host being used by the specified TCP communications stream *Stream Reference*. If a NULL pointer is passed in the *Referenced Local Port* parameter, then the local port will not be retrieved or returned by this method.

*Referenced SRTT* (support by ITK only) is a pointer to a longint variable to contain the smoother round trip time for the specified TCP communications stream *Stream Reference*. If a NULL pointer is passed in the *Referenced SRTT* parameter, then the SRTT will not be retrieved or returned by this method.

*Referenced Local IP Address* is a pointer to a longint variable to contain the IP address of the local host being used by the specified TCP communications stream *Stream Reference* . If a NULL pointer is passed in the *Referenced Local Port* parameter, then the local IP address will not be retrieved or returned by this method.

*Error Code* is the error code returned by this method. If an error occurs in this method, *Error Code* will be set to negative one (-1). If no error occurs in this method, *Error Code* will be set to zero (0).



## **TCPd\_Get\_TCP\_Info**

**TCPd\_Get\_TCP\_Info** ( *Referenced Local IP Address* ; *Referenced TCP Version Code* ; *Referenced Open Transport Version* ; *Referenced Plugin Version* ) => *Error Code*

### **TCPd\_Get\_TCP\_Info**

```
(
    -> Referenced Local IP Address : Pointer
    -> Referenced TCP Version Code : Pointer
    -> Referenced Open Transport Version : Pointer
    -> Referenced Plugin Version : Pointer
)
=> Error Code: Longint
```

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Referenced Local IP Address</i>	<b>Pointer</b>	Referenced longint variable to hold local IP address
<i>Referenced TCP Version Code</i>	<b>Pointer</b>	Referenced longint variable to hold TCP layer code
<i>Referenced Open Transport Version</i>	<b>Pointer</b>	Referenced longint variable to hold Open Transport version
<i>Referenced Plugin Version</i>	<b>Pointer</b>	Referenced longint variable to hold TCP plugin version
<i>Error Code</i>	<b>Longint</b>	Error code returned from call

The method **TCPd\_Get\_TCP\_Info** returns general information about the TCP environment on the local device and the current TCP plugin.

*Referenced Local IP Address* is a pointer to a longint variable to contain the local primary IP address. If a NULL pointer is passed in the *Referenced Local IP Address* parameter, then the local IP address will not be retrieved or returned by this method.

*Referenced TCP Version Code* is a pointer to a longint variable to contain the TCP version code of the local device. If a NULL pointer is passed in the *Referenced TCP Version Code* parameter, then the local TCP version code will not be retrieved or returned by this method. Valid values for the *Referenced TCP Version Code* are:

<u>Code</u>	<u>Description</u>
1	MacTCP v1.1.0
2	MacTCP v1.1.1
3	MacTCP v2.0.x
4	Open Transport
5	WinSock

*Referenced Open Transport Version* is a pointer to a longint variable to contain the version of Open Transport currently being used. If a NULL pointer is passed in the *Referenced Open Transport Version* parameter, then the OT version will not be retrieved or returned by this method.

*Referenced Plugin Version* is a pointer to a longint variable to contain the version of the current TCP plugin being used. If a NULL pointer is passed in the *Referenced Plugin Version* parameter, then the TCP plugin version will not be retrieved or returned by this method.

*Error Code* is the error code returned by this method. If an error occurs in this method, *Error Code* will be set to negative one (-1). If no error occurs in this method, *Error Code* will be set to zero (0).



## TCPd\_Open\_Listener



**TCPd\_Open\_Listener** ( Remote IP Address ; Remote Port ; Local Port ;  
Receive Buffer Bytes ; TCP Options ; Local IP Address ;  
Protocol ; SSL Private Key Full Path ; SSL Certificate Full Path ;  
SSL Private Key Password ; Send Timeout ) => Stream  
Reference

## **TCPd\_Open\_Listener**

```
(
    -> Remote IP Address : Longint
    -> Remote Port : Longint
    -> Local Port : Longint
    -> Receive Buffer Bytes : Longint
    -> TCP Options : Longint
    -> Local IP Address : Longint
    -> Protocol : Longint
    -> SSL Private Key Full Path : Text
    -> SSL Certificate Full Path : Text
    -> SSL Private Key Password : Text
    -> Send Timeout : Longint
)
=> Stream Reference : Longint
```

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
Remote IP Address	Longint	IP address of remote host to accept connections from
Remote Port	Longint	Port of remote host to accept connections from
Local Port	Longint	Port of local host to accept connections from
Receive Buffer Bytes	Longint	Number of bytes to use for receive buffer (ITK only)
TCP Options	Longint	MacTCP stream options (ITK only, see MacTCP Programmer's Manual for more details)
Local IP Address	Longint	IP address of local host to accept connections from (ITK only)
Protocol	Longint	Protocol to be used on stream
SSL Private Key Full Path	Text	Full path to SSL private key file (ITK v2.5.x only)
SSL Certificate Full Path	Text	Full path to SSL certificate file (ITK v2.5.x only)
SSL Private Key Password	Text	Password for SSL private key (ITK v2.5.x only)
Send Timeout	Longint	Send timeout to be used on stream (ITK only, IC is synchronous)

The method *TCPd\_Open\_Listener* opens a new TCP communications stream to listen upon using the specified parameters.

Beyond the parameters differences between the ITK and IC plugins, it is important to note this method will operate asynchronously or synchronously for TCP listens. IC does not support asynchronous TCP communications streams which listen asynchronously. So, when using the IC TCP plugin and calling this method, control will not be returned from this method until both the listener stream has been opened (assuming no errors which would be returned immediately) and a remote host has connected.

The new TCP communications stream will be added to the TCPd Streams Stack automatically within this method.

*Remote IP Address* is the IP address of the remote host to allow connections from. Set the value of this parameter to zero (0) to accept connections from any remote host.

*Remote Port* is the remote port to accept connections upon. Set the value of this parameter to zero (0) to accept connections from any remote port.

*Local Port* is the local port to listen on with the local host. If ITK v2.5.x is the current TCP plugin and *Local Port* is set to 443, the standard port used for SSL encrypted HTTP communications, then SSL will be enabled by default on the new TCP communications stream being opened for listening.

*Receive Buffer Bytes* (supported by ITK only) is the number of bytes ITK will allocate for the receive buffer on the local host. It is recommended that this value be between 8192 (8K) and 32768 (32K). If *Receive Buffer Bytes* is set to zero (0) then the default receive buffer size (8K) will be used. When IC is the current TCP plugin, this parameter is ignored.

*TCP Options* (supported by ITK only) is the MacTCP options to use for the new TCP communications stream being opened. Refer to the MacTCP Programmer's manual for details about this parameter. When using ITK v2.5.x as the TCP plugin, passing a value of 2048 will force the new TCP communications listening stream to utilize the SSL capabilities within ITK v2.5.x. When IC is the current TCP plugin, this parameter is ignored.

*Local IP Address* (supported by ITK only) is the local IP address to connect with on the local host. This is useful when the local host device is currently configured with multiple IP addresses. Setting this parameter to zero (0) will default the new TCP communications stream to listen on all of the IP addresses assigned to the local host device. When IC is the current TCP plugin, this parameter is ignored and all of the IP addresses of the local host device are used for the new TCP communications stream.

*Protocol* is the protocol to be used with the new TCP communications stream. This value has no affect on the new TCP communications stream but is stored in the TCPd Streams Stack for later reference. Valid values for the *Protocol* parameter include all of the protocol values in the constants group *TCPd\_Protocols* included with the TCP Deux component (documented separately in this manual).

*SSL Private Key Full Path* (supported by ITK v2.5.x only) is the full document path on the local machine for the SSL private key document. This parameter is needed only when the new TCP communications stream for listening has SSL enabled. When using IC or ITK v2.0.x, this parameter is ignored.

*SSL Certificate Full Path* (supported by ITK v2.5.x only) is the full document path on the local machine for the SSL certificate document. This parameter is needed only when the new TCP communications stream for listening has SSL enabled. When using IC or ITK v2.0.x, this parameter is ignored.

*SSL Private Key Password* (supported by ITK v2.5.x only) is the password used to encrypt the SSL private key when the SSL key and certificate were originally created. This parameter is needed only when the new TCP communications stream for listening has SSL enabled. When using IC or ITK v2.0.x, this parameter is ignored.

*Send Timeout* (supported by ITK only) is the default send timeout in seconds for all TCP sending on the new TCP communications stream being created. When *Send Timeout* seconds have elapsed in one of the TCP Deux send routines, the routine will return control immediately and the status of the TCP communications stream will remain the same. When IC is the current TCP plugin, this parameter is ignored.

*Stream Reference* is the unique TCP communications stream reference for the newly created TCP communications stream. This value will always be greater than zero when a valid TCP communications stream has been established. If this value is zero (0), the TCP communications stream failed to be opened properly.



## **TCPd\_Open\_Stream**

**TCPd\_Open\_Stream** (*Remote Host Name ; Remote Port ; Receive Buffer Bytes ; TCP Options ; Local Port ; Local IP Address ; Protocol* )  
=> *Stream Reference*

### **TCPd\_Open\_Stream**

```
(  
-> Remote Host Name : String [80]  
-> Remote Port : Longint  
-> Receive Buffer Bytes : Longint  
-> TCP Options : Longint  
-> Local Port : Longint  
-> Local IP Address : Longint  
-> Protocol : Longint  
)
```

=> *Stream Reference* : Longint

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Remote Host Name</i>	String [80]	Host name of remote device to connect to
<i>Remote Port</i>	Longint	Port of remote host to connect to
<i>Receive Buffer Bytes</i>	Longint	Number of bytes to assign to TCP receive buffer (ITK only)
<i>TCP Options</i>	Longint	MacTCP stream options (ITK only, see MacTCP Programmer's Manual for more details)
<i>Local Port</i>	Longint	Port on local machine to connect through (ITK only)
<i>Local IP Address</i>	Longint	IP address on local machine to connect through (ITK only)
<i>Protocol</i>	Longint	Protocol to be used for stream
<i>Stream Reference</i>	Longint	Stream reference for new TCP connection

The method *TCPd\_Open\_Stream* will open a new TCP communications stream for sending data.

The new TCP communications stream will be added to the TCPd Streams Stack automatically within this method.

To open a TCP communications stream to a specified remote host, the remote host must be listening on the specified port for the connection to be established.

*Remote Host Name* is the domain name or dotted IP address of the remote host to open a TCP communications stream with.

*Remote Port* is the remote port to connect to on the remote host. If the current TCP plugin is IC, *Remote Port* will also be the local port used to open the new TCP communications stream. If ITK v2.5.x is the current TCP plugin (with a valid ITK Pro serial) and *Remote Port* is set to 443, the standard port used for SSL encrypted HTTP communications, then SSL will be enabled by default on the new TCP communications stream being opened.

*Receive Buffer Bytes* (supported by ITK only) is the number of bytes ITK will allocate for the receive buffer on the local host. It is recommended that this value be between 8192 (8K) and 32768 (32K). If *Receive Buffer Bytes* is set to zero (0) then the default receive buffer size (8K) will be used. When IC is the current TCP plugin, this parameter is ignored.

*TCP Options* (supported by ITK only) is the MacTCP options to use for the new TCP communications stream being opened. Refer to the MacTCP Programmer's manual for details about this parameter. When using ITK v2.5.x as the TCP plugin (with a valid ITK Pro serial), passing a value of 2048 will force the new TCP communications stream to utilize the SSL capabilities within ITK v2.5.x. When IC is the current TCP plugin, this parameter is ignored.

*Local Port* (supported by ITK only) is the local port to connect with on the local host. When IC is the current TCP plugin, this parameter is ignored and the local port used is instead the same value as the *Remote Port* parameter.

*Local IP Address* (supported by ITK only) is the local IP address to connect with on the local host. This is useful when the local host device is currently configured with multiple IP addresses. Setting this parameter to zero (0) will default the new TCP communications stream to use the primary IP address assigned to the local host device. When IC is the current TCP plugin, this parameter is ignored and the primary IP address of the local host device is used for the new TCP communications stream.

*Protocol* is the protocol to be used with the new TCP communications stream. This value has no affect on the new TCP communications stream but is stored in the TCPd Streams Stack for later reference. Valid values for the *Protocol* parameter include all of the protocol values in the constants group *TCPd\_Protocols* included with the TCP Deux component (documented separately in this manual).

*Stream Reference* is the unique TCP communications stream reference for the newly created TCP communications stream. This value will always be greater than zero when a valid TCP communications stream has been established. If this value is zero (0), the TCP communications stream failed to be opened properly.



## **TCPd\_qi\_Handler\_Busy\_s**

**TCPd\_qi\_Handler\_Busy\_s** (*Handler Process ID* ) => *qi Handler Busy*

**TCPd\_qi\_Handler\_Busy\_s**

```
(
    -> Handler Process ID : Longint
)
=> qi Handler Busy: Longint
```

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Handler Process ID</i>	Longint	Process ID to lookup in TCPd Streams Stack
<i>qi Handler Busy</i>	Longint	qi for whether specified handler process ID is listed as busy in TCPd Streams Stack

The method **TCPd\_qi\_Handler\_Busy\_s** returns an indicator for whether a specified handler process ID is currently listed as being in use in the TCPd Streams Stack. A 4D handler process ID is considered "in use" merely by being listed in the TCPd Streams Stack.

*Handler Process ID* is the 4D process ID to check for existence in the TCPd Streams Stack.

*qi Handler Busy* is the indicator for whether the specified 4D handler process ID *Handler Process ID* is in use within the TCPd Streams Stack. *qi Handler Busy* will be set to zero (0) if the handler is not busy and will be set to one (1) if the handler is busy.



## TCPd\_qi\_INITed

**TCPd\_qi\_INITed** => *qi TCP Deux Initialized*

**TCPd\_qi\_INITed**

=> *qi TCP Deux Initialized: Longint*

Parameter	Type	Description
<i>qi TCP Deux Initialized</i>	Longint	qi for whether TCP Deux has been initialized successfully

The method **TCPd\_qi\_INITed** returns an indicator for whether the TCP Deux component has been initialized properly with a call to the method **INIT\_TCPd**.

*qi TCP Deux Initialized* is the indicator for whether the TCP Deux component has been initialized properly. *qi TCP Deux Initialized* will be set to zero (0) if TCP Deux has not been initialized and will be set to one (1) if TCP Deux has been initialized.



## TCPd\_Receive\_File

**TCPd\_Receive\_File** (*Stream Reference ; Local Full Path ; Filter Flag ; Receive Options ; Timeout*) => *Error Code*

**TCPd\_Receive\_File**

(  
    -> *Stream Reference* : Longint  
    -> *Local Full Path* : Text  
    -> *Filter Flag* : Longint  
    -> *Receive Options* : Longint  
    -> *Timeout* : Longint  
)  
=> *Error Code: Longint*

Parameter	Type	Description
<i>Stream Reference</i>	Longint	Stream reference to receive to file
<i>Local Full Path</i>	Text	Full path to local document to pipe stream contents to



<i>Filter Flag</i>	<b>Longint</b>	Byte filter flag to apply to received data (ITK only)
<i>Receive Options</i>	<b>Longint</b>	Receive options (e.g. leave stream open and append to existing document)
<i>Timeout</i>	<b>Longint</b>	Inactivity timeout for TCP stream while piping data to document
<i>Error Code</i>	<b>Longint</b>	Error code returned from call

The method *TCPd\_Receive\_File* will receive the data over an established TCP communications stream and save it directly to a specified document.

For data to be received over an established TCP communications stream, the stream status of the specified TCP communications stream must be "established" (8).

*Stream Reference* is the TCP communications stream reference value to send the specified data through.

*Local Full Path* is the fully qualified document path to save data received through the specified TCP communications stream *Stream Reference*.

*Filter Flag* (supported by ITK only) is an indicator code for filtering/encoding data being received. When using IC, this parameter is ignored. The following table lists the valid values for the *Filter Flag* parameter and their meanings:

<u>Value</u>	<u>Description</u>
0	no filter
1	Mac text to ISO 8859-1
2	ISO 8859-1 to Mac text
4	CR to CR/LF
8	CR/LF to CR
16	Mac text to HTML
32	HTML to Mac text
64	Mac 8 bit to HTML
128	Mac text to Base64
256	Base64 to Mac text
512	Mac text to Quoted-Printable
1024	Quoted-Printable to Mac text

*Receive Options* (supported by ITK only) is an indicator for whether the received data is to be appended to the document *Local Full Path* and whether the TCP communications stream *Stream Reference* should be released once the receiving is complete. The following table lists the valid values for this parameter and their meanings:

<u>Value</u>	<u>Description</u>
0	overwrite document contents; release stream when done
1	append to document contents; release stream when done
2	overwrite document contents; leave stream open
3	append to document contents; leave stream open

*Timeout* is number of seconds to allow for the full document contents to be received. Once *Timeout* seconds have elapsed, this method will return immediately.

*Error Code* is the error code returned by the method call in the event of error. If no error occurs, the *Error Code* value will be set to zero (0).



## **TCPd\_Receive\_to\_BLOB**

**TCPd\_Receive\_to\_BLOB** (*Stream Reference* ; *Referenced BLOB* ;  
*Maximum Receive Bytes* ; *Filter Flag* ; *Receive Options* ; *End String* ; *Timeout* ) => *Error Code*

### **TCPd\_Receive\_to\_BLOB**

```
(
    -> Stream Reference : Longint
    -> Referenced BLOB : Pointer
    -> Maximum Receive Bytes : Longint
    -> Filter Flag : Longint
    -> Receive Options : Longint
    -> End String : Text
    -> Timeout : Longint
)
```

=> *Error Code*: Longint

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Stream Reference</i>	Longint	Stream reference to receive to BLOB
<i>Referenced BLOB</i>	Pointer	Referenced BLOB to place received data into
<i>Maximum Receive Bytes</i>	Longint	Maximum number of bytes to receive into BLOB (ITK only)
<i>Filter Flag</i>	Longint	Byte filter flag to apply to received data (ITK only)
<i>Receive Options</i>	Longint	Receive options (e.g. leave stream open, append to BLOB, etc.) (ITK only)
<i>End String</i>	Text	End string value to indicate end of received data (ITK only)
<i>Timeout</i>	Longint	Inactivity timeout for TCP stream while receiving data to BLOB (ITK only)
<i>Error Code</i>	Longint	Error code returned from call

The method *TCPd\_Receive\_to\_BLOB* will receive the data over an established TCP communications stream and place it in a specified referenced BLOB.

For data to be received over an established TCP communications stream, the stream status of the specified TCP communications stream must be "established" (8).

*Stream Reference* is the TCP communications stream reference value to send the specified data through.

*Referenced BLOB* is a pointer to the BLOB to be used to place received data through the specified TCP communications stream *Stream Reference* into.

*Maximum Receive Bytes* (supported by ITK only) is the maximum number of bytes to receive over the TCP communications stream *Stream Reference*. Set the value of this parameter to zero (0) to receive all of the available data on the TCP communication stream *Stream Reference*. When using IC, this parameter is ignored.

*Filter Flag* (supported by ITK only) is an indicator code for filtering/encoding data being received. When using IC, this parameter is ignored. The following table lists the valid values for the *Filter Flag* parameter and their meanings:

<u>Value</u>	<u>Description</u>
0	no filter
1	Mac text to ISO 8859-1
2	ISO 8859-1 to Mac text
4	CR to CR/LF
8	CR/LF to CR
16	Mac text to HTML
32	HTML to Mac text
64	Mac 8 bit to HTML
128	Mac text to Base64
256	Base64 to Mac text
512	Mac text to Quoted-Printable
1024	Quoted-Printable to Mac text

*Receive Options* (supported by ITK only) is an indicator for how the receiving of data is to be terminated. When using IC, this parameter is ignored. The following table lists the valid values for this parameter and their meanings:

<u>Value</u>	<u>Description</u>
1	append to existing BLOB contents
2	do not release stream after receiving
4	look for endstring only in new received data
8	remove endstring from received data

NOTE: one or more options can be requested by summing flag values.

*End String* (supported by ITK only) is the string of text to be used to indicate the end of received data. When a string of bytes matching *End String* are received in this method, the method exits with a completed receipt of data. When using IC, this parameter is ignored.

*Timeout* (supported by ITK only) is number of ticks (1/60th of a second) to allow for the full document contents to be received. Once *Timeout* ticks have elapsed, this method will return immediately. Set the value of this parameter to zero (0) to return control immediately, or set the value to negate one (-1) to

indicate no timeout for the data being received. When using IC, this parameter is ignored.

*Error Code* is the error code returned by the method call in the event of error. If no error occurs, the *Error Code* value will be set to zero (0).



## **TCPd\_Send\_BLOB**

**TCPd\_Send\_BLOB** ( *Stream Reference* ; *Referenced BLOB* ; *Flush Flag* ; *Filter Flag* ; *Starting Offset* ; *Ending Offset* ) => *Send Buffer Bytes*

### **TCPd\_Send\_BLOB**

(  
    -> *Stream Reference* : Longint  
    -> *Referenced BLOB* : Pointer  
    -> *Flush Flag* : Longint  
    -> *Filter Flag* : Longint  
    -> *Starting Offset* : Longint  
    -> *Ending Offset* : Longint  
)  
=> *Send Buffer Bytes*: Longint

Parameter	Type	Description
<i>Stream Reference</i>	Longint	Stream reference to send BLOB on
<i>Referenced BLOB</i>	Pointer	Referenced BLOB to send
<i>Flush Flag</i>	Longint	Flag to indicate whether ITK send buffer should be flushed with this call (ITK only)
<i>Filter Flag</i>	Longint	Byte filter flag to apply to received data (ITK only)
<i>Starting Offset</i>	Longint	Offset within <i>Referenced BLOB</i> to begin sending data
<i>Ending Offset</i>	Longint	Offset within <i>Referenced BLOB</i> to stop sending data
<i>Send Buffer Bytes</i>	Longint	Number of bytes now in send buffer after completing this call

The method **TCPd\_Send\_BLOB** will send a specified portion of a referenced BLOB over an established TCP communications stream.

For data to be sent over an established TCP communications stream, the stream status of the specified TCP communications stream must be "established" (8).

*Stream Reference* is the TCP communications stream reference value to send the specified data through.

*Referenced BLOB* is a pointer to the BLOB to be used to send data through the specified TCP communications stream *Stream Reference*.

*Flush Flag* (supported by ITK only) is an indicator code for whether the ITK internal send buffer is to be used for the data being sent. If *Flush Flag* is zero (0) then the ITK send buffer will not be used; if *Flush Flag* is one (1) then the ITK send buffer will be used. When using IC, this parameter is ignored.

*Filter Flag* (supported by ITK only) is an indicator code for filtering/encoding data being sent. When using IC, this parameter is ignored. The following table lists the valid values for the *Filter Flag* parameter and their meanings:

<u>Value</u>	<u>Description</u>
0	no filter
1	Mac text to ISO 8859-1
2	ISO 8859-1 to Mac text
4	CR to CR/LF
8	CR/LF to CR
16	Mac text to HTML
32	HTML to Mac text
64	Mac 8 bit to HTML
128	Mac text to Base64
256	Base64 to Mac text
512	Mac text to Quoted-Printable
1024	Quoted-Printable to Mac text

*Starting Offset* is the starting offset within the BLOB *Referenced BLOB* to begin sending data from. Set *Starting Offset* to zero (0) to indicate the beginning of *Referenced BLOB*.

*Ending Offset* is the ending offset within the BLOB *Referenced BLOB* for the last character to send data from. Set *Ending Offset* to MAXLONG (the native 4D constant) to indicate the ending of *Referenced BLOB*.

*Send Buffer Bytes* is the error code returned by the method call in the event of error. If no error occurs, the *Send Buffer Bytes* value will be set to the number of bytes in the ITK output buffer or zero (0) if the ITK output buffer is not being used or IC is the current TCP plugin.



## **TCPd\_Send\_File**

**TCPd\_Send\_File** ( *Stream Reference* ; *Local Full Path* ; *Filter Flag* ; *Sending Block Size* ; *Starting Offset* ; *Ending Offset* ; *Send Options* ) => *Error Code*

### **TCPd\_Send\_File**

```
(
  -> Stream Reference : Longint
  -> Local Full Path : Text
  -> Filter Flag : Longint
  -> Sending Block Size : Longint
  -> Starting Offset : Longint
  -> Ending Offset : Longint
  -> Send Options : Longint
)
=> Error Code: Longint
```

Parameter	Type	Description
<i>Stream Reference</i>	Longint	Stream reference to send document on
<i>Local Full Path</i>	Text	Full path to local document to send
<i>Filter Flag</i>	Longint	Byte filter flag to apply to sent data (ITK only)
<i>Sending Block Size</i>	Longint	Maximum transmission block size to use for sending data
<i>Starting Offset</i>	Longint	Offset within <i>Local Full Path</i> document contents to begin sending data

<i>Ending Offset</i>	<b>Longint</b>	<b>Offset within <i>Local Full Path</i> document contents to stop sending data</b>
<i>Send Options</i>	<b>Longint</b>	<b>Indicator for which file fork of document to send</b>
<i>Error Code</i>	<b>Longint</b>	<b>Error code returned from call</b>

The method *TCPd\_Send\_File* will send the contents of a specified document over an established TCP communications stream.

For data to be sent over an established TCP communications stream, the stream status of the specified TCP communications stream must be "established" (8).

*Stream Reference* is the TCP communications stream reference value to send the specified data through.

*Local Full Path* is the fully qualified document path to be sent through the specified TCP communications stream *Stream Reference*.

*Filter Flag* (supported by ITK only) is an indicator code for filtering/encoding data being sent. When using IC, this parameter is ignored. The following table lists the valid values for the *Filter Flag* parameter and their meanings:

<u>Value</u>	<u>Description</u>
0	no filter
1	Mac text to ISO 8859-1
2	ISO 8859-1 to Mac text
4	CR to CR/LF
8	CR/LF to CR
16	Mac text to HTML
32	HTML to Mac text
64	Mac 8 bit to HTML
128	Mac text to Base64
256	Base64 to Mac text
512	Mac text to Quoted-Printable
1024	Quoted-Printable to Mac text

*Sending Block Size* (supported by ITK only) is the data block size to use for progressively sending the file contents through the specified TCP communications



stream. When using the IC TCP plugin, this parameter is ignored.

*Starting Offset* is the starting offset within the document *Local Full Path* to begin sending data from. Set *Starting Offset* to zero (0) to indicate the beginning of *Local Full Path*.

*Ending Offset* is the ending offset within the document *Local Full Path* for the last character to send data from. Set *Ending Offset* to MAXLONG (the native 4D constant) to indicate the ending of *Local Full Path*.

*Send Options* (supported by ITK only) is an indicator flag used on Macintosh to indicate which fork of the document *Local Full Path* to send. If *Send Options* is zero (0) then the data fork is sent; if *Send Options* is one (1) then the resource fork is sent. If the current TCP plugin is IC, then this parameter is ignored and the data fork of the document *Local Full Path* only is sent.

*Error Code* is the error code returned by the method call in the event of error. If no error occurs, the *Error Code* value will be set to zero (0).



## TCPd\_Send\_Text

**TCPd\_Send\_Text** ( *Stream Reference* ; *Send Text* ; *Flush Flag* ; *Filter Flag* )  
=> *Send Buffer Bytes*

**TCPd\_Send\_Text**  
(  
-> *Stream Reference* : **Longint**  
-> *Send Text* : **Text**  
-> *Flush Flag* : **Longint**  
-> *Filter Flag* : **Longint**  
)  
=> *Error Code* : **Longint**

Parameter	Type	Description
<i>Stream Reference</i>	<b>Longint</b>	Stream reference to send text on

<i>Send Text</i>	Text	Text to send
<i>Flush Flag</i>	Longint	Flag to indicate whether ITK send buffer should be flushed with this call (ITK only)
<i>Filter Flag</i>	Longint	Byte filter flag to apply to sent data (ITK only)
<i>Send Buffer Bytes</i>	Longint	Error code returned from call

The method *TCPd\_Send\_Text* will send a specified text value over an established TCP communications stream.

For data to be sent over an established TCP communications stream, the stream status of the specified TCP communications stream must be "established" (8).

*Stream Reference* is the TCP communications stream reference value to send the specified text through.

*Send Text* is the text to send through the specified TCP communications stream *Stream Reference*.

*Flush Flag* (supported by ITK only) is an indicator code for whether the ITK internal send buffer is to be used for the data being sent. If *Flush Flag* is zero (0) then the ITK send buffer will not be used; if *Flush Flag* is one (1) then the ITK send buffer will be used. When using IC, this parameter is ignored.

*Filter Flag* (supported by ITK only) is an indicator code for filtering/encoding data being sent. When using IC, this parameter is ignored. The following table lists the valid values for the *Filter Flag* parameter and their meanings:

<u>Value</u>	<u>Description</u>
0	no filter
1	Mac text to ISO 8859-1
2	ISO 8859-1 to Mac text
4	CR to CR/LF
8	CR/LF to CR
16	Mac text to HTML
32	HTML to Mac text
64	Mac 8 bit to HTML

128	Mac text to Base64
256	Base64 to Mac text
512	Mac text to Quoted-Printable
1024	Quoted-Printable to Mac text

*Send Buffer Bytes* is the error code returned by the method call in the event of error. If no error occurs, the *Send Buffer Bytes* value will be set to the number of bytes in the ITK output buffer or zero (0) if the ITK output buffer is not being used or IC is the current TCP plugin.



## **TCPd\_Set\_ProcessID\_by\_Index\_s**

**TCPd\_Set\_ProcessID\_by\_Index\_s** (*Stream Stack Index* ; *Handler Process ID* )

**TCPd\_Set\_ProcessID\_by\_Index\_s**  
(  
    -> *Stream Stack Index* : Longint  
    -> *Handler Process ID* : Longint  
)

<b>Parameter</b>	<b>Type</b>	<b>Description</b>
<i>Stream Stack Index</i>	Longint	Index into TCPd Streams Stack
<i>Handler Process ID</i>	Longint	Handler process ID to set into TCPd Streams Stack

The method *TCPd\_Set\_ProcessID\_by\_Index\_s* sets the 4D handler process ID in the TCPd Streams Stack matching the specified TCPd Streams Stack index.

*Stream Stack Index* is the row index to set in the TCPd Streams Stack.

*Handler Process ID* is the 4D handler process ID value to set in the TCPd Streams Stack for the row matching the TCPd Streams Stack row index *Stream Stack Index* .



## TCPd\_Set\_Status\_by\_Index\_s

**TCPd\_Set\_Status\_by\_Index\_s** ( *Stream Stack Index* ; *Stream Status* )

**TCPd\_Set\_Status\_by\_Index\_s**

```
(
    -> Stream Stack Index : Longint
    -> Stream Status : Longint
)
```

Parameter	Type	Description
<i>Stream Stack Index</i>	Longint	Index into TCPd Streams Stack
<i>Stream Status</i>	Longint	Stream status to set into TCPd Streams Stack

The method **TCPd\_Set\_Status\_by\_Index\_s** sets the TCP communications stream status in the TCPd Streams Stack matching the specified TCPd Streams Stack index.

*Stream Stack Index* is the row index to set in the TCPd Streams Stack.

*Stream Status* is the TCP communications stream status value to set in the TCPd Streams Stack for the row matching the TCPd Streams Stack row index *Stream Stack Index* .



## TCPd\_Update\_Statuses\_by\_Prot\_s

**TCPd\_Update\_Statuses\_by\_Prot\_s** ( *Protocol* )

**TCPd\_Update\_Statuses\_by\_Prot\_s**

```
(
    -> Protocol : Longint
)
```

Parameter	Type	Description
<i>Protocol</i>	Longint	Protocol to update all stream status values for within TCPd Streams Stack

The method *TCPd\_Update\_Statuses\_by\_Prot\_s* updates the TCP communications stream statuses in the TCPd Streams Stack for all rows matching a specified protocol value. Stream statuses are sequentially checked by calling the current TCP plugin for each row in the TCPd Streams Stack matching the specified protocol value.

*Protocol* is the protocol value in the TCPd Streams Stack to update all TCP communications streams statuses for . Valid values for the *Protocol* parameter include all of the protocol values in the constants group *TCPd\_Protocols* included with the TCP Deux component (documented separately in this manual).



## TCPd\_Wait\_for\_NotStatus

**TCPd\_Wait\_for\_NotStatus** (*Stream Reference* ; *Stream Status to Not Await* ; *Timeout* ) => *Stream Status*

**TCPd\_Wait\_for\_NotStatus**  
 (  
     -> *Stream Reference* : Longint  
     -> *Stream Status to Not Await* : Longint  
     -> *Timeout* : Longint  
 )  
 => *Stream Status*: Longint

Parameter	Type	Description
<i>Stream Reference</i>	Longint	Stream reference to wait for
<i>Stream Status to Not Await</i>	Longint	Status value to wait until specified stream reference does not equal
<i>Timeout</i>	Longint	Timeout in seconds to wait for specified stream reference status conditions to be met
<i>Stream Status</i>	Longint	Last status of stream before returning from call

The method *TCPd\_Wait\_for\_NotStatus* waits for a specified stream reference to not equal a particular stream status within a specified period of time.

This method is ideal for use when waiting for a TCP communications stream to begin changing state while waiting for a connection to be established or for a closing to occur.

This method will wait for the conditions to be met on the specified stream reference by accessing the current TCP plugin for the stream status. This method will automatically keep the TCPd Streams Stack synchronised with the current state of the TCP communications stream status retrieved from the current TCP plugin.

*Stream Reference* is the stream reference of the TCP communications stream to be checking.

*Stream Status to Not Await* is the stream status value to await for the stream *Stream Reference* to not be equal to. Valid values for the parameter *Stream Status to Not Await* are:

Status	Name	Description
0	Closed	no connection
2	Listen	listening for incoming connection
4	SYN received	incoming connection being established
6	SYN sent	outgoing connection being established
8	Established	connection up
10	FIN Wait 1	connection up; close sent
12	FIN Wait 2	connection up; close sent and acknowledged
14	Close Wait	connection up; close received
16	Closing	connection up; close issued and received
18	Last ACK	connection up; close issued and received
20	Time Wait	connection being broken

*Timeout* is the number of seconds this method is to wait for the stream *Stream Reference* to no longer have a stream status of *Stream Status to Not Await*. If the *Timeout* seconds elapse without the stream status conditions being met, this method will return control immediately to the calling method.

*Stream Status* is the status of the TCP communications stream *Stream Reference* when the return conditions, whether they be by timeout or not, have been met within this method.



## TCPd\_Wait\_for\_Status

**TCPd\_Wait\_for\_Status** ( *Stream Reference* ; *Stream Status to Await* ;  
Timeout ) => *Stream Status*

### TCPd\_Wait\_for\_Status

```
(  
    -> Stream Reference : Longint  
    -> Stream Status to Await : Longint  
    -> Timeout : Longint  
)  
=> Stream Status: Longint
```

Parameter	Type	Description
<i>Stream Reference</i>	Longint	Stream reference to wait for
<i>Stream Status to Await</i>	Longint	Status value to wait until specified stream reference does equal
<i>Timeout</i>	Longint	Timeout in seconds to wait for specified stream reference status conditions to be met
<i>Stream Status</i>	Longint	Last status of stream before returning from call

The method *TCPd\_Wait\_for\_Status* waits for a specified stream reference to equal a particular stream status within a specified period of time.

This method is ideal for use when waiting for a TCP communications stream to reach a particular state while waiting for a connection to be established or for a closing to occur.

This method will wait for the conditions to be met on the specified stream reference by accessing the current TCP plugin for the stream status. This method will automatically keep the TCPd Streams Stack synchronised with the current state of the TCP communications stream status retrieved from the current TCP plugin.

*Stream Reference* is the stream reference of the TCP communications stream to be checking.

*Stream Status to Await* is the stream status value to await for the stream *Stream Reference* to be equal to. Valid values for the parameter *Stream Status to Await* are:

Status	Name	Description
0	Closed	no connection
2	Listen	listening for incoming connection
4	SYN received	incoming connection being established
6	SYN sent	outgoing connection being established
8	Established	connection up
10	FIN Wait 1	connection up; close sent
12	FIN Wait 2	connection up; close sent and acknowledged
14	Close Wait	connection up; close received
16	Closing	connection up; close issued and received
18	Last ACK	connection up; close issued and received
20	Time Wait	connection being broken

*Timeout* is the number of seconds this method is to wait for the stream *Stream Reference* to have a stream status of *Stream Status to Await* . If the *Timeout* seconds elapse without the stream status conditions being met, this method will return control immediately to the calling method.

*Stream Status* is the status of the TCP communications stream *Stream Reference* when the return conditions, whether they be by timeout or not, have been met within this method.





# Version History

The following is a brief version history of the TCP Deux component. It details release notes, bug fixes, and changes for each version publicly available.

# TCP Deux v1.0.0b01

*released 20010517*

## Changes:

First public beta release of the component.



## Using TCP Deux

asd



## TCP Deux Error Codes

asd





## Index

asd