TCP Deux

Developer Documentation

v1.1.1

Published and Distributed Worldwide by Deep Sky Technologies, Inc.

Deep Sky Technologies, Inc.
P.O. Box 6897
Vero Beach, FL 32961-6897
772.794.9494

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 custom or commercial databases or applications created by the original licensee. No additional product license or royalty is required.

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 it's 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, it's 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

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 Client Deux, POP3 Client Deux, FTP Client Deux and HTTP Client Deux, among others. Details and documentation for these components are provided separately.

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.

Mr. James A. 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

TCP Deux is a 4th Dimension component which provides cross-platform TCP wrappers for different TCP plugins available for 4th Dimension. The plugins supported within TCP Deux include 4D Internet Commands v6.7.x, 4D Internet Commands v6.8.x, 4D Internet Commands v7.0.x (4D IC 2003), Internet ToolKit v2.0.x, and Internet ToolKit v2.5.x.

With the release of TCP Deux v1.1.0 and above, compatibility with 4D v6.8.x is now available. TCP Deux v1.1.1 and above added support for 4D v7.0.x (4D 2003). Since 4D v6.8.x and 4D v7.0.x (4D 2003) is a full carbon version of 4th Dimension, support for Internet ToolKit v2.0.x is no longer an option. TCP Deux will automatically check which version of 4D is running and will allow only compatible versions of the listed plugins to be used with the individual versions of 4th Dimension.

With the TCP Deux component, a 4th Dimension developer can code TCP level routines without worrying about which particular plugin is going to be used. TCP Deux provides a very simple upgrade path for developers to write one set of code. With TCP Deux, a 4D developer can begin by using the free 4D Internet Commands for all TCP communications and upgrade to the commercial Internet ToolKit plugin when the needs arises, all without changing a single line of code.

Separate 4D components for higher level TCP protocols are available from Deep Sky Technologies, Inc. This includes components for SMTP, POP3, FTP client, HTTP/HTTPS client, and more. Of course, we are always looking for suggestions about what other high level TCP protocols you need, so we can write the components to handle those as well!

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. With the release of TCP Deux v1.1.0 and above, compatibility with 4D v6.8.x is now supported. This includes running TCP Deux on MacOS 8/9, MacOS X, and Windows 98/2000/NT/XP. TCP Deux v1.1.1 and above includes support for 4D v7.0.x (4D 2003) and continues support of all available platforms for deployment of 4D with TCP Deux.

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 protocols and servers 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.

Installing 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 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.

On Macintosh, the Affix TCP Deux document is entitled **Affix_TCP_Deux.4DX** (under 4D v6.8.x and 4D v7.0.x, there is now a carbonized version entitled **Affix_TCP_Deux.4CX**) 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: **Affix_TCP_Deux.4DX** and **Affix_TCP_Deux.RSR**. These two document 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 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. A plugin stub has been provided for ITK v2.5.x to reduce the total software package size which is required.

Please read 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 and On Server Startup database methods.

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, 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 of 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's 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 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.

The 4D Internet Commands plugin (entitled **4D_IC_v673.4DX** for 4D v6.7.x and **4D_IC_v682.4DX** for 4D v6.8.x, and **4D_IC_v700mc01.4DXB** for 4D v7.0.x, respectively) is automatically initialized by the TCP Deux component when initialized. So, there is no need to initialize the TCP layer from within your code. Basically, this consists of a call to **IT_MacTCPInit** which is handled within the TCP Deux component.

The 4D Internet Commands plugins are provided unmodified directly from 4D, Inc., and 4D SA.

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 (compatible with 4D v6.7.x only), install the ITK v2.0.x version of the plugin contained within the TCP Deux software package. This version is entitled **ITKp_v204.4DX**.

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, you must still install the 4D Internet Commands plugin for compilation compatibility.

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_v260.4DX**.

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_s260.4DX**.

Note: if your application will be using ITK v2.0.x for network connectivity, you must still install the 4D Internet Commands plugin for compilation compatibility.

4D v6.8.x

With the availability of 4D v6.8.x, the complete 4th Dimension environment is now fully carbonized. 4D as a carbonized application allows for a single set of tools to function on either MacOS X or MacOS v9.x using CarbonLib.

With the release of 4D v6.8.x, there is now a new plugin architecture available for third party developers. As well, there are some changes which have been made in the actually plugin hierarchy and naming conventions. There have also been changes made to the component architecture within the 4D product line. Reading the release notes for 4D v6.8.x is a great source of information regarding these changes.

TCP Deux is currently available with compatibility for 4D v6.8.x. This comes in the form of new affix and plugin documents for use with 4D v6.8.x. The TCP Deux archive contains both 4D v6.7.x and 4D v6.8.x compatible versions of the component, affix documents, and plugins.

When updating an existing database from 4D v6.7.x to 4D v6.8.x, we have found that installed components will no longer update properly. The first time that a component is updated after upgrading a 4D structure, the component must first be removed from the structure before installing the new version of the component. We have not seen any problems with doing this other than the extra step required to remove the component using 4D Insider.

It is also important that you install the correct affix and plugin documents for your environment. Whether you are running under MacOS X or MacOS 9 is not a factor. Rather, whether you are using 4D v6.7.x or 4D v6.8.x is the determining factor. Copy the appropriate documents for your current version of 4D from the component archive for use in your 4D structure. The differences between the 4D v6.7.x and 4D v6.8.x affix and plugin documents is simple to determine; the names of the documents compatible with 4D v6.7.x end in “.4DX” and the names of the documents compatible with 4D v6.8.x end in “.4CX”.

4D v7.0.x (4D v2003)

TCP Deux is currently available with compatibility for 4D v7.0.x. This comes in the form of new affix and plugin documents for use with 4D v7.0.x. The TCP Deux archive contains 4D v6.7.x, 4D v6.8.x, and 4D v7.0.x compatible versions of the component, affix documents, and plugins.

It is also important that you install the correct affix and plugin documents for your environment. Whether you are running under MacOS X or MacOS 9 is not a factor. Rather, whether you are using 4D v6.7.x, 4D v6.8.x, or 4D v7.0.x is the determining factor. Copy the appropriate plugin documents for your current version of 4D from the component archive for use in your 4D structure.

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 from your 4DX folder.

Remove the internet connectivity plugins from your 4DX folder if they are no longer needed.

Remove the call to the method *INIT_TCPd* from your On Startup and On Server Startup database methods.

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...

Updating to TCP Deux v1.1.1

Updating to the latest release of the TCP Deux component is a simple procedure. Follow the instructions contained in the section **“Installing TCP Deux” on page 10** to update the code within the structure.

In particular for TCP Deux, make certain that the latest release of the Affix TCP Deux document is placed in the 4DX folder.

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 protocols and servers 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:

Field Name	Type
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 man-

ual 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:

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

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. "deep-skytech.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 TCP Deux, IP addresses for local and remote hosts are handled as longint values. This provides a convenient and memory efficient means for handling IP addresses. When an IP address is required for a specific parameter in a TCP Deux method, it will need to be a longint value by default.

The BASh component package contains routines to convert IP addresses between longint values and 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.

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

Dotted IP Address	Longint IP Address
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

Dotted IP Address	Longint IP Address
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 utitlized 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:

Constant	Value
TCPd_IC_v67x_Plugin	1
TCPd_ITK_v25x_Plugin	2
TCPd_ITK_v20x_Plugin	4
TCPd_IC_v68x_Plugin	8
TCPd_IC_v70x_Plugin	16

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:

Constant	Value
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_DNS_Listen	9
TCPd_DNS_Session	10
TCPd_SMPP_Listen	11
TCPd_SMPP_Session	12
TCPd_HTTPS_Listen	301
TCPd_HTTPS_Session	302
TCPd_other_Listen	601
TCPd_other_Session	602
TCPd_other_SSL_Listen	801
TCPd_other_SSL_Session	802

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

Code Modules

All of the code within the TCP Deux component is organized into modules. Each module is designated by a three (3) to five (5) character module prefix. All of the module prefixes are used within the name of every object within the module (methods names, variable names, semaphore names, etc.). This allows for the easy identification of any object within the TCP Deux component.

Each module contains a set of methods which can be used throughout your database once the TCP Deux component is installed. Method names all begin with the module prefix followed by an underscore ("_") characters. The remainder of the method name then describes the function of the method.

TCPd Module

The TCPd module handles all aspects of the TCP Deux component at this time.

ENV_Get_TCPd_HardName_Long

ENV_Get_TCPd_HardName_Long => *Long Hard Name*

ENV_Get_TCPd_HardName_Long
=> *Long Hard Name :* Text

	Parameter	Type	Description
	Long Hard Name	Text	Full, hard coded name of TCP Deux component including versioning information

The method ***ENV_Get_TCPd_HardName_Long*** returns the full, hard coded name of the TCP Deux component, including versioning information.

Long Hard Name is the full, hard coded name of the TCP Deux component. As of this release, this will always return the value “TCP_Deux_v1.1.1”.

Note: this method was added as of TCP Deux v1.0.1.

ENV_Get_TCPd_HardName_Short

ENV_Get_TCPd_HardName_Short => *Short Hard Name*

ENV_Get_TCPd_HardName_Short
=> *Short Hard Name :* Text

	Parameter	Type	Description
	Short Hard Name	Text	Short hard coded name of TCP Deux component

The method ***ENV_Get_TCPd_HardName_Short*** returns the short hard coded name of the TCP Deux component.

Short Hard Name is the shortened, hard coded name of the TCP Deux component. As of this release, this will always return the value "TCP_Deux".

Note: this method was added as of TCP Deux v1.0.1.

INIT_TCPd

INIT_TCPd *(Plugin to Use ; Macintosh Plugin Serial ; Windows Plugin Serial ; TCP Deux Serial { ; qi Initialize 4D IC Plugin })*

INIT_TCPd

(

-> *Plugin to Use :* Longint
-> *Macintosh Plugin Serial :* Text
-> *Windows Plugin Serial :* Text
-> *TCP Deux Serial :* Text
{ -> *qi Initialize 4D IC Plugin :* Longint }

)

	Parameter	Type	Description
	Plugin to Use	Longint	TCP plugin to use with TCP Deux; values are available in the TCP Deux constants group TCPd_Plugin_Types
	Macintosh Plugin Serial	Text	Macintosh or primary ITK v2.5.x plugin serial
	Windows Plugin Serial	Text	Windows or ITK SSL Server v2.5.x plugin serial
	TCP Deux Serial	Text	TCP Deux serial
	qi Initialize 4D IC Plugin	Longint	Code for whether 4D IC plugin should be initialized when not default plugin for use by TCPd

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 ITK SSL Server serial should be passed in this parameter. For IC, this parameter can be left empty.

TCP Deux Serial is the TCP Deux serial which came with your purchase of the TCP Deux component. 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 an empty serial number; this will allow for 30 minutes of unlimited use.

qi Initialize 4D IC Plugin is an optional parameter to indicate whether the 4D Internet Commands plugin should be initialized when it is not the default TCP plugin to be used with TCP Deux. When any version of ITK is the plugin to be used by TCP Deux, this parameter can be used to stop the initialization of the 4D IC plugin. By default, the 4D IC plugin is always intialized. Passing a value of zero (0) for this parameter will force TCP Deux to not initialize the 4D IC plugin. Passing a value of one (1) for this parameter will force TCP Deux to always initialze the 4D IC plugin in all situations.

RES_Open_TCPd

RES_Open_TCPd => *Resource Fork File Reference*

RES_Open_TCPd
=> *Resource Fork File Reference :* Time

	Parameter	Type	Description
	Resource Fork File Reference	Time	File reference to resource fork of TCP Deux Affix document

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_NoWait

TCPd_Close_Stream_NoWait *(Stream Reference)*

TCPd_Close_Stream_NoWait
(
-> *Stream Reference :* Longint
)

	Parameter	Type	Description
	Stream Reference	Longint	Reference for the TCP stream to close

The method ***TCPd_Close_Stream_NoWait*** will close an open TCP stream without waiting for confirmation of the closing.

Note: this routine will not release the TCP stream, but only closes the stream. A call to the method ***TCPd_Release_Stream*** should follow a call to this method so that the TCP stream is also released from memory.

Stream Reference is the stream identifier for the stream to be closed. It is the same value as returned by the methods ***TCPd_Open_Stream*** and ***TCPd_Open_Listener***.

Note: this method was added as of TCP Deux v1.0.1.

TCPd_Close_Streams_by_Protocol

TCPd_Close_Streams_by_Protocol *(Protocol)*

TCPd_Close_Streams_by_Protocol
(
-> *Protocol :* Longint
)

	Parameter	Type	Description
	Protocol	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 *Protocol* include all of the protocol values in the constants group TCPd_Protocols.

TCPd_CloseRelease_Stream

TCPd_CloseRelease_Stream *(Stream Reference)*

TCPd_CloseRelease_Stream
(
-> *Stream Reference :* Longint
)

	Parameter	Type	Description
	Stream Reference	Longint	TCP stream reference to close

The method ***TCPd_CloseRelease_Stream*** will close and release 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 and released is removed once the stream has completed closing and been released.

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

Note: as of TCP Deux v1.0.1, this method has been renamed from ***TCPd_Close_Stream*** to ***TCPd_CloseRelease_Stream*** .

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
)

	Parameter	Type	Description
	Referenced Stream References	Pointer	Pointer to longint array to hold TCP stream references
	Referenced Ports	Pointer	Pointer to longint array to hold ports
	Referenced Protocols	Pointer	Pointer to longint array to hold protocols
	Referenced Statuses	Pointer	Pointer to longint array to hold TCP stream statuses
	Referenced IP Addresses	Pointer	Pointer to longint array to hold remote IP addresses
	Referenced Handler Process IDs	Pointer	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 is 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 is 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 is 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 is 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 is 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 is 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
	Stream Stack Row Count	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

)

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

The method ***TCPd_ERROR*** acts as a callback method from within the ARR module for errors that may occur. Any time an error condition is detected within the ARR module, 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 call 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
	Domain Name	Text	Domain name to lookup
	Mode	Longint	Mode indicator for which IP address to return (ITK only)
	IP Address	Longint	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 *Domain Name*. The following table lists the valid values for the *Mode* parameter and the functionality associated with each value:

Mode	Action
- 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 *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
	IP Address	Longint	IP address to lookup
	Mode	Longint	Mode indicator for domain name format to return
	Domain Name	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 *Mode* 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_s *(Stream Reference)* => *Stream Stack Index*

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

	Parameter	Type	Description
	Stream Reference	Longint	Stream reference to lookup in TCPd Streams Stack
	Stream Stack Index	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_IPAddress_by_Index_s

TCPd_Get_IPAddress_by_Index_s *(Streams Stack Index)* => *IP Address*

TCPd_Get_IPAddress_by_Index_s
(
-> *Streams Stack Index :* Longint
)
=> *IP Address : Longint*

	Parameter	Type	Description
	Streams Stack Index	Longint	Index into TCPd Streams Stack
	IP Address	Longint	IP address within specified index of TCPd Streams Stack

The method ***TCPd_Get_IPAddress_by_Index_s*** returns the IP address 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.

IP Address is the IP address stored in the row specified by *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 IP Address will be set to negative one (-1).

Note: this method was added as of TCP Deux v1.0.1.

TCPd_Get_IPAddress_s

TCPd_Get_IPAddress_s *(Stream Reference) => IP Address*

TCPd_Get_IPAddress_s
(
-> *Stream Reference* : Longint
)
=> *IP Address : Longint*

	Parameter	Type	Description
	Stream Reference	Longint	Stream reference to lookup in TCPd Streams Stack
	IP Address	Longint	IP address listed in TCPd Streams Stack for specified stream reference

The method ***TCPd_Get_IPAddress_s*** returns the IP address 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.

IP Address is the IP address 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, IP Address will be set to negative one (-1).

Note: this method was added as of TCP Deux v1.0.1.

TCPD_Get_Index_by_ProcessID_s

TCPD_Get_Index_by_ProcessID_s *(Process ID) => Streams Stack Index*

TCPD_Get_Index_by_ProcessID_s

(

-> *Process ID :* Longint

)

=> *Streams Stack Index :* Longint

	Parameter	Type	Description
	Process ID	Longint	ID of process to search for within TCPd Streams Stack
	Streams Stack Index	Longint	Index in TCPd Streams Stack matching *Process ID*

The method ***TCPd_Get_Index_by_ProcessID_s*** returns the index in the **TCPd Streams Stack** matching the specified process ID provided.

Process ID is the process ID to look for in the TCPd Streams Stack.

Streams Stack Index is the index into the TCPd Streams Stack matching *Process ID*. If *Process ID* is invalid, out of range, or not found in the TCPd Streams Stack, then *Streams Stack Index* will be set to negative one (-1).

Note: this method was added as of TCP Deux v1.0.1.

TCPD_Get_LocalPort_by_Index_s

TCPD_Get_LocalPort_by_Index_s *(Streams Stack Index)* => *Local Port*

TCPD_Get_LocalPort_by_Index_s
(
-> *Streams Stack Index :* Longint
)
=> *Local Port :* Longint

	Parameter	Type	Description
	Streams Stack Index	Longint	Index into TCPd Streams Stack
	Local Port	Longint	Local port within specified index of TCPd Streams Stack

The method ***TCPd_Get_LocalPort_by_Index_s*** returns the local port 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.

Local Port is the local port stored in the row specified by*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 *Local Port* will be set to negative one (-1).

Note: this method was added as of TCP Deux v1.0.1.

TCPd_Get_LocalPort_s

TCPd_Get_LocalPort_s *(Stream Reference)* => *Port*

TCPd_Get_LocalPort_s
(
-> *Stream Reference :* Longint
)

=> *Port :* Longint

	Parameter	Type	Description
	Stream Reference	Longint	Stream reference to lookup in TCPd Streams Stack
	Port	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 *Stream Reference* is not found in the TCPd Streams Stack, *Port* will be set to negative one (-1).

TCPD_Get_ProcessID_by_Index_s

TCPD_Get_ProcessID_by_Index_s *(Streams Stack Index)* => *Process ID*

TCPD_Get_ProcessID_by_Index_s

(

-> *Streams Stack Index :* Longint

)

=> *Process ID :* Longint

	Parameter	Type	Description
	Streams Stack Index	Longint	Index into TCPd Streams Stack
	Process ID	Longint	Process ID within specified index of TCPd Streams Stack

The method ***TCPd_Get_ProcessID_by_Index_s*** returns the process ID 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.

Process ID is the process ID stored in the row specified by *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 *Process ID* will be set to negative one (-1).

Note: this method was added as of TCP Deux v1.0.1.

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
	Stream Reference	Longint	Stream reference to lookup in TCPd Streams Stack
	Handler Process ID	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_by_Index_s

TCPD_Get_Protocol_by_Index_s *(Streams Stack Index)* => *Protocol*

TCPD_Get_Protocol_by_Index_s
(
-> *Streams Stack Index :* Longint
)
=> *Protocol :* Longint

	Parameter	Type	Description
	Streams Stack Index	Longint	Index into TCPd Streams Stack
	Protocol	Longint	Protocol within specified index of TCPd Streams Stack

The method ***TCPd_Get_Protocol_by_Index_s*** returns the protocol 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.

Protocol is the protocol stored in the row specified by *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 *Protocol* will be set to negative one (-1). Valid values for *Protocol* include all of the protocol values in the constants group TCPd_Protocols included with the TCP Deux component.

Note: this method was added as of TCP Deux v1.0.1.

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
	Protocol	Longint	Protocol to lookup in TCPd Streams Stack
	Streams Count	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 *Protocol* include all of the protocol values in the constants group TCPd_Protocols included with the TCP Deux component.

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
	Stream Reference	Longint	Stream reference to lookup in TCPd Streams Stack
	Protocol	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 *Stream Reference* is not found in the TCPd Streams Stack, *Protocol* will be set to negative one (-1). Valid values for *Protocol* include all of the protocol values in the constants group TCPd_Protocols included with the TCP Deux component.

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
	Stream Reference	Longint	Stream reference to check status of directly from TCP plugin
	Selector Code	Longint	ITK status selector value (ITK only, see ITK manual for ITK_TCPStatus)
	Stream Status	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 *Stream Status*:

Status	Name	Description
-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_by_Index_s

TCPD_Get_Status_by_Index_s *(Streams Stack Index) => Stream Status*

TCPD_Get_Status_by_Index_s

(

-> *Streams Stack Index :* Longint

)

=> *Stream Status :* Longint

	Parameter	Type	Description
	Streams Stack Index	Longint	Index into TCPd Streams Stack
	Stream Status	Longint	Status value for specified stream reference returned directly from TCPd Streams Stack

The method ***TCP_Get_Status_by_Index_s*** returns the stream status in the **TCPd Streams Stack** matching the specified TCPd Streams Stack index 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 Stack Index is the row index to look for in the TCPd Streams Stack.

Stream Status is the stream status value in the TCPd Streams Stack matching the specified TCPd Streams Stack index provided in *Stream Stack Index*. If the value for *Stream Stack Index* is invalid, out of range, or 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*:

Status	Name	Description
-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

Status	Name	Description
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

Note: this method was added as of TCP Deux v1.0.1.

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
	Stream Reference	Longint	Stream reference to check status of directly from TCPd Streams Stack
	Stream Status	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 com-

munications 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 *Stream Status*:

Status	Name	Description
-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

(

-> *Streams Stack Index :* Longint

)

=> *Stream Reference :* Longint

	Parameter	Type	Description
	Streams Stack Index	Longint	Index into TCPd Streams Stack
	Stream Reference	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 *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

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

	Parameter	Type	Description
	Referenced Local IP Address	Pointer	Referenced longint variable to hold local IP address
	Error Code	Longint	Error code returned from call

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 for *Referenced Remote IP Address*, 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 if passed for *Referenced Remote Port*, 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 for *Referenced Local Port*, 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 smoothed round trip time for the specified TCP communications stream *Stream Reference*. If a NULL pointer is passed for *Referenced SRTT*, 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

	Parameter	Type	Description
	Referenced Local IP Address	Pointer	Referenced longint variable to hold local IP address
	Referenced TCP Version Code	Pointer	Referenced longint variable to hold TCP layer code
	Referenced Open Transport Version	Pointer	Referenced longint variable to hold Open Transport version
	Referenced Plugin Version	Pointer	Referenced longint variable to hold TCP plugin version
	Error Code	Longint	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 for *Referenced Local IP Address*, 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 for *Referenced TCP Version Code*, then the local TCP version code will not be retrieved or returned by this method. Valid values for *Referenced TCP Version Code* are:

Code	Description
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 for *Referenced Open Transport Version*, 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 for *Referenced Plugin Version* , 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
```

	Parameter	Type	Description
	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 (IC v6.8.1 or above or ITK v2.5.x only)
	SSL Certificate Full Path	Text	Full path to SSL certificate file (IC v6.8.1 or above or ITK v2.5.x only)
	SSL Private Key Password	Text	Password for SSL private key (IC v6.8.1 or above or ITK v2.5.x only)
	Send Timeout	Longint	Send timeout to be used on stream (ITK only, IC v6.8.1 or above does not accept this parameter, IC v6.8.0 or below is synchronous)
	Stream Reference	Longint	Stream reference for new TCP listener

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 IC v6.8.1 or above or 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 *Local IP Address* 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 *Protocol* include all of the protocol values in the constants group TCPd Protocols included with the TCP Deux component.

SSL Private Key Full Path (supported by IC v6.8.1 or above or 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 below version 6.8.1 or ITK v2.0.x, this parameter is ignored.

SSL Certificate Full Path (supported by IC v6.8.1 or above or ITK v2.5.x only) iis 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 below version 6.8.1 or ITK v2.0.x, this parameter is ignored.

SSL Private Key Password (supported by IC v6.8.1 or above or 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 below 6.8.1 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, *Send Timeout* 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 *Stream Reference* 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

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

	Parameter	Type	Description
	Protocol	Longint	Protocol to be used for stream
	Stream Reference	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 commucations 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 IC v6.8.1 or above or 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 plu-

gin, this parameter is ignored and the local port used is instead the same value as *Remote Port*.

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 *Local IP Address* 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 *Protocol* include all of the protocol values in the constants group TCPd Protocols included with the TCP Deux component.

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 *Stream Reference* 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

	Parameter	Type	Description
	Handler Process ID	Longint	Process ID to lookup in TCPd Streams Stack

	Parameter	Type	Description
	qi Handler Busy	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_IC

TCPd_qi_IC => *qi Using IC*

TCPd_qi_IC
=> *qi Using IC* : Longint

	Parameter	Type	Description
	qi Using IC	Longint	Indicator for whether TCP Deux is currently set to use the 4D Internet Commands plugin

The method ***TCPd_qi_IC*** returns qi for whether the TCP Deux component is currently set to use the 4D Internet Commands plugin for all TCP communications.

qi Using IC is a longint value returned to indicate whether 4D Internet Commands is the TCP plugin currently set to be used by TCP Deux. This value will be set to zero (0) if 4D Internet Commands is not set to be used and will be set to one (1) if 4D Internet Commands is set to be used.

Note: the method ***TCPd_qi_IC*** was added in TCP Deux v1.1.1.

TCPd_qi_INITed

TCPd_qi_INITed => *qi TCP Deux Initialized*

TCPd_qi_INITed
=> *qi TCP Deux Initialized* : Longint

	Parameter	Type	Description
	qi TCP Deux Initialized	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_qi_ITK

TCPd_qi_ITK => *qi Using ITK*

TCPd_qi_ITK
=> *qi Using ITK* : Longint

	Parameter	Type	Description
	qi Using ITK	Longint	Indicator for whether TCP Deux is currently set to use the Internet ToolKit plugin

The method ***TCPd_qi_ITK*** returns qi for whether the TCP Deux component is currently set to use the Internet ToolKit plugin for all TCP communications.

qi Using ITK is a longint value returned to indicate whether Internet ToolKit is the TCP plugin currently set to be used by TCP Deux. This value will be set to zero (0) if Internet Tool-Kit is not set to be used and will be set to one (1) if Internet ToolKit is set to be used.

Note: the method ***TCPd_qi_ITK*** was added in TCP Deux v1.1.1.

TCPd_qi_Plugin_w_SSLStreams

TCPd_qi_Plugin_w_SSLStreams *=> qi SSL Support*

TCPd_qi_Plugin_w_SSLStreams
=> qi SSL Support : Longint

	Parameter	Type	Description
	qi SSL Support	Longint	Indicator for whether current TCP plugin supports SSL connections

The method ***TCPd_qi_Plugins_w_SSLStreams*** returns an indicator for whether the current TCP plugin in use by TCP Deux component suppports SSL connections. IC v6.8.1 or above and ITK v2.5.x support SSL; IC v6.8.0 or below and ITK v2.0.x do not support SSL.

qi SSL Support is longint value returned to indicate whether the TCP plugin currently set to be used by the TCP Deux component supports SSL connections. This value will be set to zero (0) if SSL connections are not supported and will be set to one (1) if SSL connections are supported.

Note: the method ***TCPd_qi_Plugins_w_SSLStreams*** was added in TCP Deux v1.1.1.

TCPd_qi_Protocol_w_SSL

TCPd_qi_Protocol_w_SSL *(Protocol)=> qi SSL*

TCPd_qi_Protocol_w_SSL

(

-> *Protocol :* Longint

)

=> *qi SSL* : Longint

	Parameter	Type	Description
	Protocol	Longint	TCPd constant for protocol to check for SSL use
	qi SSL	Longint	Indicator for whether SSL is used in specified protocol

The method ***TCPd_qi_Protocol_w_SSL*** returns an indicator for whether a specified protocol used SSL

Protocol is a longint value to signify the protocol to check for use of SSL. The values supported come directly from the TCPd_Protocols constants group and should be used when passing this parameter to this routine.

qi SSL is a longint indicator returned to signify whether the specified protocol uses SSL. This value will be set to zero (0) if the specified protocol does not use SSL and will be set to one (1) if the specified protocol does use SSL.

Note: the method ***TCPd_qi_Protocol_w_SSL*** was added in TCP Deux v1.1.1.

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
	Stream Reference	Longint	Stream reference to receive to file
	Local Full Path	Text	Full path to local document to pipe stream contents to
	Filter Flag	Longint	Byte filter flag to apply to received data (ITK only)
	Receive Options	Longint	Receive options (e.g. leave stream open and append to existing document)
	Timeout	Longint	Inactivity timeout for TCP stream while piping data to document
	Error Code	Longint	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 *Filter Flag* and their meanings:

Value	Description
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

Value	Description
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 *Receive Options* and their meanings:

Value	Description
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, *Error Code* 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
```

	Parameter	Type	Description
	Stream Reference	Longint	Stream reference to receive to BLOB
	Referenced BLOB	Pointer	Referenced BLOB to place received data into
	Maximum Receive Bytes	Longint	Maximum number of bytes to receive into BLOB (ITK only)
	Filter Flag	Longint	Byte filter flag to apply to received data (ITK only)
	Receive Options	Longint	Receive options (e.g. leave stream open, append to BLOB, etc.) (ITK only)
	End String	Text	End string value to indicate end of received data (ITK only)
	Timeout	Longint	Inacitivty timeout for TCP stream while receiving data to BLOB (ITK only)
	Error Code	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 *Filter Flag* and their meanings:

Value	Description
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 *Receive Options* and their meanings:

Value	Description
1	append to existing BLOB contents
2	do not release stream after receiving
4	look for endstring only in new received data

Value	Description
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 *Timeout* to zero (0) to return control immediately, or set the value to negatize 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, *Error Code* will be set to zero (0).

TCPD_Release_Stream

TCPD_Release_Stream *(Stream Reference ; Stop Listening Code)*

TCPD_Release_Stream
(
-> *Stream Reference :* Longint
-> *Stop Listening Code :* Longint
)

	Parameter	Type	Description
	Stream Reference	Longint	Reference for the TCP stream to release
	Stop Listening Code	Longint	Code value for whether listener stream should stop listening on port (valid for ITK only)

The method ***TCPd_Release_Stream*** will release an open TCP stream or listener and remove its entry from the TCPd Streams Stack.

Note: this routine will not close the TCP stream, but only releases the stream. A call to the method ***TCPd_Close_Stream_NoWait*** should preceed a call to this method so that the TCP stream is closed before releasing it from memory.

Stream Reference is the stream identifier for the stream to be released. It is the same value as returned by the methods ***TCPd_Open_Listener*** and ***TCPd_Open_Stream***.

Stop Listening Code is a value for indicating whether the internal listener should stop listening on the port which is the stream Stream Reference is on. If *Stop Listening Code* is set to zero (0) then the internal listener will not stop listening; if *Stop Listening Code* is set to one (1) then the internal listener will stop listening. This value is valid only when using ITK; when using IC, this value will be ignored.

Note: this method was added as of TCP Deux v1.0.1.

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
	Stream Reference	Longint	Stream reference to send BLOB on

	Parameter	Type	Description
	Referenced BLOB	Pointer	Referenced BLOB to send
	Flush Flag	Longint	Flag to indicate whether ITK send buffer should be flushed with this call (ITK only)
	Filter Flag	Longint	Byte filter flag to apply to received data (ITK only)
	Starting Offset	Longint	Offset within *Referenced BLOB* to begin sending data
	Ending Offset	Longint	Offset within *Referenced BLOB* to stop sending data
	Send Buffer Bytes	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 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 *Filter Flag* and their meanings:

Value	Description
0	no filter

Value	Description
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 *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 *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
	Stream Reference	Longint	Stream reference to send document on
	Local Full Path	Text	Full path to local document to send
	Filter Flag	Longint	Byte filter flag to apply to sent data (ITK only)
	Sending Block Size	Longint	Maximum transmission block size to use for sending data
	Starting Offset	Longint	Offset within *Local Full Path* document contents to begin sending data
	Ending Offset	Longint	Offset within *Local Full Path* document contents to stop sending data
	Send Options	Longint	Indicator for which file fork of document to send
	Error Code	Longint	Error code returned from call

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 *Filter Flag* and their meanings:

Value	Description
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
)
=> *Send Buffer Bytes :* Longint

	Parameter	Type	Description
	Stream Reference	Longint	Stream reference to send text on
	Send Text	Text	Text to send
	Flush Flag	Longint	Flag to indicate whether ITK send buffer should be flushed with this call (ITK only)
	Filter Flag	Longint	Byte filter flag to apply to sent data (ITK only)
	Send Buffer Bytes	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 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 *Filter Flag* and their meanings:

Value	Description
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, *Send Buffer Bytes* 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

(

-> *Streams Stack Index :* Longint

-> *Handler Process ID :* Longint

)

	Parameter	Type	Description
	Streams Stack Index	Longint	Index into TCPd Streams Stack
	Handler Process ID	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

(

-> *Streams Stack Index :* Longint

-> *Stream Status :* Longint

)

	Parameter	Type	Description
	Streams Stack Index	Longint	Index into TCPd Streams Stack
	Stream Status	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
	Protocol	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 *Protocol* include all of the protocol values in the constants group TCPd_Protocols included with the TCP Deux component.

TCPd_Update_Statuses_by_Type_s

TCPd_Update_Statuses_by_Type_s *(Stream Type)*

TCPd_Update_Statuses_by_Type_s

(

-> *Stream Type :* Longint

)

	Parameter	Type	Description
	Stream Type	Longint	Type of stream to update all stream status values for within TCPd Streams Stack

The method TCPd_Update_Statuses_by_Type_s updates the TCP communications stream statuses in the TCPd Streams Stack for all rows matching a specified stream type. Stream statuses are sequentially checked by calling the current TCP plugin for each row in the TCPd Streams Stack matching the specified stream type.

Stream Type is the type of stream in the TCPd Streams Stack to update all TCP communications streams statuses. A *Stream Type* value of zero (0) will update the status for all sessions; a *Stream Type* value of one (1) will update the status for all listeners; a *Stream Type* value of two (2) will update the statuses of all streams, whether they be session or listener streams.

Note: this method was added as of TCP Deux v1.0.1.

TCPd_Wait_for_NotStatus

TCPd_Wait_for_NotStatus *(Stream Reference ; Stream Status to Not Await ; Timeout) => Stream Status*

TCPd_Wait_for_NotStatus

(

-> *Streams Reference :* Longint

-> *Stream Status to Not Await :* Longint

-> *Timeout :* Longint

)

=> *Stream Status :* Longint

	Parameter	Type	Description
	Stream Reference	Longint	Stream reference to wait for
	Stream Status to Not Await	Longint	Status value to wait until specified stream reference does not equal
	Timeout	Longint	Timeout in seconds to wait for specified stream reference status conditions to be met
	Stream Status	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 *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

Status	Name	Description
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 *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
(
-> *Streams Reference :* Longint
-> *Stream Status to Await :* Longint
-> *Timeout :* Longint
)
=> *Stream Status :* Longint

	Parameter	Type	Description
	Stream Reference	Longint	Stream reference to wait for
	Stream Status to Await	Longint	Status value to wait until specified stream reference does equal

	Parameter	Type	Description
	Timeout	Longint	Timeout in seconds to wait for specified stream reference status conditions to be met
	Stream Status	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 *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

Status	Name	Description
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 *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 v1.1.1

released March 17th, 2003

Changes:

Fixed rare bug in which some 4D serials and names could cause component to not serialize properly.

Corrected resource ID conflicts within platforms of BASh Affix documents; made certain all affix documents coincide with the new plugin ID for TCP Deux (29601).

Made Affix document on Mac under 4D v67x a true plugin stub.

Corrected issue with bug in 4D v68x on Windows in which resource names can cause conflicts.

Made all component methods invisible.

Corrected bug in ***ENV_Get_TCPd_RF_FullPath*** method in which it would scan the system 4DX folder even if the Affix document was found in the 4DX folder next to the structure.

Renumbered values for constants TCPd_HTTPS_Listen and TCPd_HTTPS_Stream from 101 and 102 respectively to 301 and 302.

Added stream control type support for IC v681 and above.

Added support for new parameters for later version of 4D IC, specifically IC v681 and above.

Added check to method ***TCPd_Open_Stream*** for whether IC plugin supports SSL streams and whether specified protocol requires SSL.

Added TCPd_IC_v70x_Plugin constant to TCPd_Plugin_Types constants group for support of 4D v70x.

Added support for 4D v70x throughout component code.

Added distribution of 4D v70x compatible versions of component and plugins.

Added 'cfrg' resource to all affix documents for compatibility with 4D v70x.

Added the protected methods:
TCPd_qi_Plugin_w_SSLStreams
TCPd_qi_Protocol_w_SSL

Changed methods from private to protected:
TCPd_qi_IC
TCPd_qi_ITK

Added constants to constants group TCPd_Protocols:
TCPd_DNS_Listen
TCPd_DNS_Session
TCPd_other_SSL_Listen
TCPd_other_SSL_Session
TCPd_SMPP_Listen
TCPd_SMPP_Session

TCP Deux v1.1.0

released February 28th, 2002

Changes:

Correct bug in serialization checking routines in which the compiled state of the application was not being properly checked against allowable serialization.

Increased minimum version of BASh compatible with this release to v1.7.0.

Added support for new 4D plugin architecture under 4D v6.8.x in which plugins can be located next to the current application in any environment.

Dropped support for ITK v2.0.x from component when using 4D v6.8.x or above.

Added new plugin type constant, TCPd_IC_v68x_Plugin (8), for use with 4D v6.8.x.

Added compatibility check for different version of 4D Internet Commands against 4D APPL version; an error will be generated if the wrong version of 4D IC is being used with the version of the 4D APPL being used.

Added support for 4D Internet Commands v6.8.x as the plugin type when calling the method TCPd_qi_IC.

Added optional parameter to INIT_TCPd to indicate whether 4D Internet Commands should be initialized when it is not being used as the TCPd plugin of choice; by default, this is done now in case this parameter is not passed.

Changed the name of the Mac affix document to "Affix_TCP_Deux.4DX".

Changed the name of the Win affix document to "Affix_TCP_Deux".

Changed the name of the Mac 4D Internet Commands plugin to "4D_IC_v673.4DX".

Changed the name of the Win 4D Internet Commands plugin to "4D_IC_v673".

Created a carbon affix document entitled "Affix_TCP_Deux.4CX".

Dropped completely the 4D Internet Commands plugin stub and instead only distribute the full version of 4D IC (the only savings this used to afford was a smaller size, but this was also added confusion for many developers).

Included a carbon 4D Internet Commands plugin entitled "4D_IC_v680f01.4DX".

Updated ITK to v2.5.1b04, a very stable version of ITK v2.5.x containing important bug fixes in the TCP stream API.

TCP Deux v1.0.3

released 20020104

Changes:

Changed name of affix document on Windows from "Afx_TCPd" to "Affix_TCP_Deux".

Changed **TCPd Streams Stack** locking and unlocking methods to now be index based, thereby speeding considerably access into the stack.

Changed all calls to locking and unlocking **TCPd Streams Stack** to now be index based, thereby speeding considerably access into the stack.

Changed ***TCPd_Send_BLOB*** routine to protected and the wrapper routines called to private, as they were inadvertently left open in the last release.

Updated both serial checking routines to pull data from BASh's ENV module, speeding these routines up significantly.

Fixed a bug in the checking of serial numbers when initializing the component.

Updated 4D Internet Commands (i.e. 4DIC or IC) to v6.7.3.

TCP Deux v1.0.2

released 20011115

Changes:

Added ability to send MAXLONG as ending offset to end of BLOB within the method ***TCPd_Send_BLOB***.

Fixed a bug in the method ***TCPd_Send_BLOB*** when using IC in which the specified offsets within the BLOB were not being respected when the BLOB was sent.

Added the ability for an empty serial to be used as a demo serial; this allows for 30 minutes of unlimited use of TCP Deux.

TCP Deux v1.0.1

released 20010906

Changes:

Modified the method ***TCPd_Get_Status_s***; removed the locking of the TCPd Streams Stack when getting the status from the stack.

Modified the routine ***TCPd_Get_Status***; removed the code that gets the actual status from the stream with the new method ***TCPd_Get_Status_Direct*** that gets the status directly from the stream.

Modified the routine ***TCPd_Update_Statuses_by_Prot_s***; now the **TCPd Streams Stack** is locked before looping through the stack while it updates all of the statuses with the new method ***TCPd_Get_Status_Direct*** and unlocks it when completed.

Updated documentation for ***TCPd_Close_Stream*** to make it clear that this method not only closes a TCP stream but also releases it.

Renamed the method ***TCPd_Close_Stream*** to ***TCPd_CloseRelease_Stream***.

Removed the locking and unlocking of the TCPd Streams stack from the methods ***TCPd_Get_LocalPort_s***, ***TCPd_Get_ProcessID_s***, ***TCPd_Get_Protocol_s***, and ***TCPd_Get_StreamRef_by_Index_s***.

Changed the method ***TCPd_Open_Listener*** to get the stream status directly from the plugin using the new method ***TCPd_Get_Status_Direct*** when adding the new stream to the **TCPd Streams Stack**.

Optimized all routines for increased speed when using ITK as the plugin.

Fixed a bug in which ITK v2.0.x was not recognized; this could cause all forms of problems when running with ITK v2.0.x as the TCP plugin.

Changed the serialization checking routines to take advantage of the new calls within BASh v1.6.0; these changes decrypt the serials for TCPd using the BLOB routines in the CRYPT module, which have complete compatibility with double byte operating systems (thanks to Ken Ishimoto of K's Room for help with this).

Modified the routine ***TCPd_Receive_to_BLOB*** to remove any checks on the stream status before receiving over the stream; under certain conditions when using ITK on Windows, a TCP stream could be closed while there is still data to be received into the TCP buffer for the stream in the plugin.

Changed the following private methods into protected methods for use by developers:

ENV_Get_TCPd_HardName_Long
ENV_Get_TCPd_HardName_Short
TCPd_qi_IC
TCPd_qi_ITK

Added the methods:

TCPd_Close_Stream_NoWait
TCPd_Get_IPAddress_by_Index_s
TCPd_Get_IPAddress_s
TCPd_Get_LocalPort_by_Index_s
TCPd_Get_Plugin_in_Use
TCPd_Get_ProcessID_by_Index_s
TCPd_Get_Protocol_by_Index_s
TCPd_Get_Status_by_Index_s
TCPd_Get_Status_Direct (private method)
TCPd_Release_Stream
TCPd_Update_Statuses_by_Type_s

TCP Deux v1.0.0

released 20010725

Changes:

First full release of the TCP Deux component. No changes were made since TCP Deux v1.0.0b03.

Errors

The listing of error codes and conditions is obviously a continuously updated process. With practically every new version of TCP Deux, the error codes and conditions can and do change. Though it has been a long time coming, we are now documenting much of the error conditions that can occur in TCP Deux.

Different methods in the TCP Deux component can generate errors when used incorrectly or when used under the wrong circumstances. When an error condition is encountered, the methods within the TCP Deux component will call the applicable "_ERROR" method. The "_ERROR" methods are documented above. The first parameter sent to an "_ERROR" method is the error code identifying the unique error condition that occurred.

With future versions of the TCP Deux component (and other components to come), the management and handling of error conditions will become much better documented, much clearer to understand, and easier to handle in your code. For now, though, reading this manual thoroughly is the best single source of understanding for the error conditions that can arise in using the TCP Deux component.

Error Codes

When an "_ERROR" method is called in the TCP Deux component, the first parameter is always an error code. This error code is always a 7 digit integer indicating the unique error condition which was detected in the code.

These 7 digit numbers actually consist of two pieces for uniquely identifying the error code and condition. The first five digits is an internal code for the module which an error occured within. The last two digits identifies the unique error condition from within a module that has been detected.

The following is a quick listing of all of the error codes, grouped by the module which the error codes are from. Each error code includes the textual description of the error condition.

TCPd: 29020

01	Stream reference does not exist.
02	TCP plugin not chosen.
03	Internet ToolKit serial number(s) invalid.
04	TCP is not active. Some components may not be active.
05	4D Internet Commands error.
06	Expected a pointer.
07	Referenced value is not the correct type.
08	ITK SSL error.
09	TCP stream was not closed.
10	TCP stream is still established after close has been issued.
11	TCP protocol not valid.
12	Invalid stream reference from a TCP listen or TCP open.
13	Index out of range in TCP stack.
14	TCPp module has not been initialized.
15	Failed to lock TCPd stack.
16	Failed to mark TCPd module as initialized.
17	Stream reference already exists in stack.
18	Attempting to send an empty value.
19	TCPp module not serialized or demo period has expired.
20	Invalid range within BLOB specified.
21	Failed to initialize 4D Internet Commands plugin.
22	Internet ToolKit v2.0.x is compatible with 4D v6.7.x only.
23	Incompatible version of 4D Internet Commands being used.
24	Current plugin does not support SSL.

Method Listing

The following is a listing of all of the methods within the TCP Deux component (COMPILER methods are not included in this listing), including all private methods which are not directly available in the API when the TCP Deux component is installed. Following each method is a list of the error codes which each method can generate.

ENV_Get_TCPd_HardName_Long
ENV_Get_TCPd_HardName_Short
ENV_Get_TCPd_RF_FullPath
INIT_TCPd
RES_Open_TCPd
TCPd_Add_s
    2902014
    2902015
    2902017
TCPd_Check_Serial_Full
TCPd_Check_Serial_Quick
TCPd_CloseRelease_Stream
    2902002
    2902005
    2902009
    2902010
    2902014
TCPd_Close_Streams_by_Protocol
    2902011
    2902014
TCPd_Close_Stream_NoWait
    2902002
    2902005
    2902009
    2902014
TCPd_Copy_s
    2902015
TCPd_Count_Rows_s
TCPd_Delete_s
    2902001
    2902014
    2902015
TCPd_ERROR
TCPd_Get_DNSLookUp
    2902002
    2902005
    2902014
TCPd_Get_DNSLookUp_Reverse
    2902002
    2902005
    2902014
TCPd_Get_Index_by_ProcessID_s
    2902014
TCPd_Get_Index_s
    2902014
TCPd_Get_IPAddress_by_Index_s
    2902013
    2902014

TCPd_Get_IPAddress_s
    2902014
TCPd_Get_LocalPort_by_Index_s
TCPd_Get_LocalPort_s
    2902014
TCPd_Get_Plugin_in_Use
TCPd_Get_ProcessID_by_Index_s
TCPd_Get_ProcessID_s
    2902014
TCPd_Get_Protocol_by_Index_s
TCPd_Get_Protocol_Count_s
    2902011
    2902014
TCPd_Get_Protocol_s
    2902014
TCPd_Get_Status
    2902014
TCPd_Get_Status_by_Index_s
TCPd_Get_Status_Direct
    2902002
    2902005
    2902014
TCPd_Get_Status_s
    2902014
TCPd_Get_StreamRef_by_Index_s
TCPd_Get_Stream_Information
    2902002
    2902014
TCPd_Get_TCP_Info
    2902002
    2902005
    2902006
    2902007
    2902014
TCPd_INIT
    2902002
    2902004
    2902019
    2902021
    2902022
    2902023
TCPd_Lock_Stack_s
    2902014
    2902015
TCPd_Open_Listener
    2902002
    2902005
    2902012
    2902014
    2902019
TCPd_Open_Stream
    2902002
    2902005
    2902012
    2902014
    2902019
    2902024
TCPd_qi_Handler_Busy_s
    2902014

TCPd_qi_IC
	2902014
TCPd_qi_INITed
TCPd_qi_ITK
	2902014
TCPd_qi_Plugin_w_SSLStreams
TCPd_qi_Protocol_w_SSL
TCPd_Receive_File
	2902002
	2902014
TCPd_Receive_to_BLOB
	2902002
	2902005
	2902014
TCPd_Release_Stream
	2902002
	2902009
	2902014
TCPd_Send_BLOB
	2902002
	2902005
	2902014
	2902018
	2902020
TCPd_Send_File
	2902002
	2902005
	2902014
TCPd_Send_Text
	2902002
	2902005
	2902014
	2902018
TCPd_Set_ProcessID_by_Index_s
	2902013
	2902014
	2902015
TCPd_Set_Status_by_Index_s
	2902013
	2902014
	2902015
TCPd_Unlock_Stack_s
	2902014
TCPd_Update_Statuses_by_Prot_s
	2902014
	2902015
TCPd_Update_Statuses_by_Type_s
	2902014
	2902015
TCPd_Wait_For_NotStatus
	2902014
TCPd_Wait_For_Status
	2902014