TCP Server Deux

Developer Documentation

v1.0.0

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

Make certain that you are using the latest releases of the required components and plugins with TCP Server Deux.

Other optional components which can be used with TCP Server Deux include HTTP Server Deux. Details and documentation for these components are provided separately.

Acknowledgements

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

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

Later tweaks and additions to the TCP Server Deux component have resulted from the training of James A. Crate. Mr. Crate's experience in many different programming environments has provided refreshing insights into the overall structure and organization of the core routines at DSTi, the same core routines which are available in the BASh and TCP Server Deux components.

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

Features

TCP Server Deux is a 4th Dimension component which provides a solid foundation to build a server for any TCP/IP protocol (e.g. HTTP, FTP, DNS, etc.) in 4th Dimension. TCP Server Deux also requires that TCP Deux is used with Internet ToolKit v2.0.x or v2.5.x.

With the TCP Server Deux component, a 4th Dimension developer can build a server by just writing the code for the actions that need to be performed when a connection to the server is made. The developer will not have to worry about how to properly build and manage the server code.

A separate 4D component that implements an HTTP server is available from Deep Sky Technologies, Inc. HTTP Server Deux performs all the functions that are necessary in a web server; the developer will only need to write the code to actually build the HTML. Of course, we are always looking for suggestions about other TCP server protocols you need, so we can write the components to handle those as well!

System Requirements

The TCP Server 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 Server Deux v1.0.0 and above, compatibility with 4D v6.8.x is now supported. This includes running TCP Server Deux on MacOS 8/9, MacOS X, and Windows 98/2000/NT/XP.

As of TCP Server Deux v1.0.0, there are now some minimum requirements for lower level components used by TCP Server Deux. In particular, you must be using BASh v.1.7.0 or above and TCP Deux v1.1.0 or above. Previous versions of BASh and TCP Deux will not function properly with TCP Server Deux v1.0.0 and above, resulting in errors when TCP Server Deux is initialized.

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 Server 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 Server Deux

Installing TCP Server Deux or updating an existing version of TCP Server Deux within a 4D database is performed using 4D Insider. The activity primarily consists of installing the TCP Server Deux component in a database structure opened with 4D Insider (installing the TCP Server 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 Server Deux component and the limited number of object names, conflicts should be very rare.

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

Open the uncompiled structure that you wish to install TCP Server 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 Server Deux component file and click on the "Open" button.

4D Insider parses the TCP Server 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 Server Deux component.

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

In the event of an update, 4D Insider compares the version numbers of both the currently installing TCP Server Deux component and the already installed TCP Server 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 Server Deux component and adds new objects from the new TCP Server 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 Server Deux component, 4D Insider will prompt you with an appropriate dialog box.

Save the database in 4D Insider.

Call the method INIT_TCPsd early in the On Startup and On Server Startup database methods.

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

Details about the ***INIT_TCPsd*** method can be found in the method documentation, below.

The TCP Server 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 Server 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 Server 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 Server 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 Server 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 Server 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 Server Deux.

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

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 Server Deux is currently available with compatibility for 4D v6.8.x. The TCP Server Deux archive contains both 4D v6.7.x and 4D v6.8.x compatible versions of the component.

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.

Since 4D v6.8.x is still currently in beta, it is best to report any issues with components immediately to Deep Sky Technologies, Inc. We can research the issue very quickly and notify 4D, Inc./4D SA of any compatibility and functional problems quickly so that the final release of 4D v6.8.x is as bug free as possible.

Uninstalling TCP Server Deux

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

To uninstall TCP Server Deux from your 4D database:

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

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

Consider again how great the TCP Server 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 call to the method *INIT_TCPsd* from your On Startup and On Server Startup database methods.

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

Updating to TCP Server Deux v1.0.0

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

TCP and TCP Server Deux Conventions

Throughout this manual, and all other documentation and supporting materials, included with the TCP Server 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 Server 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 Server 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.

TCPsd Services

TCP Server Deux uses a concept called **services**. A **service** is a set of TCP listening connections for a specific IP address and a specific port. For instance, you may want to accept HTTP requests on port 80, which is the standard HTTP port. You may also want to accept SSL layered http requests on port 443, which is the standard SSL port for web requests. And finally, you may want to accept HTTP requests on port 8888 for your secret administration interface. These are three different services, each with specific information for that service.

If that sounds easy enough, you are probably wondering how you work with these services once they are started. Well, that is pretty simple too. When a connection is received by the TCPsd Server, it runs through three phases: Before, During, and After. In each phase, it executes the method that you have specified when you created the service. After the connection is completed, the Post Processor Handler is also called.

The three phases are convenient, because in our example, we have three services all receiving HTTP requests. The process of getting and parsing the request is the same for all three services, as is the sending of data back to the web browser. The code for getting and parsing the request can be executed in the Before phase, and the code for sending the data back to the web browser can be executed in the After phase. Those methods will probably be used in the Before and After phase of each service. The During phase of each service could use a different method, one which is appropriate to handle those specific types of request.

The Post Processor Handler method is a little different than the other handler methods, though. The Post Processor Handler will be called after the TCP stream is already marked and in the process of closing. Within the Post Processor Handler, no TCP information can be retrieved from the current stream. The Post Processor is designed to handle common maintenance and logging routines which are not dependent upon the TCP stream being available.

Since TCP Server Deux uses TCP Deux, you will need to use TCP Deux methods to send and receive data in your handler methods. However, since you are using TCP Server Deux, you will be freed from the connection and process management that is crucial to a high performance TCP server.

And, keep in mind, with the implementation of services within the TCP Server Deux component, it is a simple matter to implement a single 4D application that handles serving of multiple protocols; for instance, it

is not unreasonable to write an integrated email and DNS server that has a web interface for administration.

TCPsd Services Stack

The TCPsd Services Stack is an internal mechanism within the TCP Server Deux component to efficiently keep track of the active servers. This allows you to set up multiple servers. For instance, you could set up both an HTTP server and an FTP server in your database, and TCP Server Deux can manage both of them at the same time.

The TCPsd Services Stack can only be changed when the TCP Server is not running. Methods are provided to start and stop TCP Server Deux, so you can change your server configuration without having to restart your entire database application.

The TCPsd Services Stack is basically just a listing of data about each TCP server implemented with the TCP Server Deux component. For each TCP server managed by the TCP Server Deux component, there is one row in the TCPsd Services Stack; uniqueness within the TCP Services Stack is determined by a combination of the IP address(es) the server is listening on and the local port. Each row in the TCPsd Services Stack contains a single field for each of the following pieces of information:

Field Name	Type
Service Name	String [32]
IP Address	Longint
Local Port	Longint
Listening Streams	Longint
Server Protocol	Longint
Before Handler Method Name	String [32]
During Handler Method Name	String [32]
After Handler Method Name	String [32]
Post Processor Handler Method Name	String [32]

The service name is a unique, non-empty name used to reference each TCP Server.

The IP Address specifies the local IP address to listen on. This allows you to set up multiple servers on different IP address. You can also listen on all addresses available on the local host or only the primary IP address on the local host.

The Local Port is the local port to be used for the TCP communications.

The Listening Streams is the number of listening streams to be used in this server.

The Server Protocol is the coded TCP Deux protocol value to be used on the TCP communications stream. See the section **TCPd_Protocols** in this manual for details about the different coded values, available as constants in the TCP Deux component, available for the protocols field.

The Handler method names specify which method will be executed during the phases of the connection. Typically, code to parse the request could be run in the Before handler. Code to process the request could be run in the During handler. Code to send the result could be run in the After handler. Of course, you may only want to use one or two of these handlers; there is no restrictions to passing empty values for any or all of the handler method names.

The Post Processor Handler method is a little different than the other handler methods, though. The Post Processor Handler will be called *after* the TCP stream is already marked and in the process of closing. Within the Post Processor Handler, no TCP information can be retrieved from the current stream. The Post Processor is designed to handle common maintenance and logging routines which are not dependent upon the TCP stream being available.

Time Log

The Time Log is functionality included within the TCP Server Deux component to help determine the request and response handling times of different steps within the TCP Server Deux component and within your code. In general, the Time Log is a dump of the amount of time taken in handling a TCP stream at different steps in the handling process.

The Time Log can be enabled and disabled at any time during the operation of a 4D application. Methods are available within the TCP Server Deux component for turning the Time Log on and off at any time during the running of your 4D application.

The Time Log dump is appending to a document created next to the current 4D structure document. This document contains a single line for each connection established through the TCP Server Deux component. As well, each time the Time Log is enabled or disabled, a new line noting the action taken and the current date and time is written to the Time Log document. Each time the Time Log is enabled, a new header line is written to the Time Log document, too.

Header lines and enabling and disabling lines that are written to the Time Log document always begin with an exclamation character "!". All other entries for handled connections are merely tab delimited values as outlined below.

A connection entry made to the Time Log document consists of the following tab delimited entries:

Value Title	Value Units
Connection Established	DTS
Before Before Method	Milliseconds
Before During Method	Milliseconds
Before After Method	Milliseconds
After After Method	Milliseconds
Close Issued	Milliseconds
Release Issued	Milliseconds

The first value in each Time Log entry is the full DTS (Date Time Stamp) of when the TCP connection was established. All subsequent values are the number of milliseconds between the current entry and the previous entry. This format allows for a simple visual scan to be done on a Time

Log document to identify average response times for different steps within the TCP Server Deux component and within your handler code.

The Time Log document, created in the same directory as the current 4D structure document, is entitled "TCPsd_TimeLog.txt". If this document already exists when the Time Log is enabled, entries are appended to the existing document. If the Time Log document does not exist when the Time Log is enabled, the document will be automatically created.

IP Addresses

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

The BASh component package contains routines to quickly and easily convert between IP addresses stored as longint values and IP addresses stored as string values (dotted IP addresses). The routines ***CONV_IP_to_Longint*** and ***CONV_Longint_to_IP*** allow for the conversion of a single value of one type to another.

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

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
63.175.177.37	1068478757

Constants

There are no custom constants included with the TCP Server Deux component package. However, TCP Server Deux makes use of constants from the TCP Deux component package, which is required for TCP Server Deux. 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 and TCP Server Deux.

TCPd_Protocols

The TCPd_Protocols constants group, available within the TCP Deux component, 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_HTTPS_Listen	9
TCPd_HTTPS_Session	10
TCPd_other_Listen	601
TCPd_other_Session	602

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 Server 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 Server Deux component.

Each module contains a set of methods which can be used throughout your database once the TCP Server 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.

TCPsd Module

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

ENV_Get_TCPsd_HardName_Long

ENV_Get_TCPsd_HardName_Long => *Long Hard Name*

ENV_Get_TCPsd_HardName_Long
=> *Long Hard Name :* Text

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

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

Long Hard Name is the full, hard coded name of the TCP Server Deux component. As of this release, this will always return the value “TCP_Server_Deux_v1.0.0”.

ENV_Get_TCPsd_HardName_Short

ENV_Get_TCPsd_HardName_Short => *Short Hard Name*

ENV_Get_TCPsd_HardName_Short
=> *Short Hard Name :* Text

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

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

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

INIT_TCPsd

INIT_TCPd *(TCP Server Deux Serial)*

INIT_TCPd

(

-> *TCP Server Deux Serial :* Text

)

	Parameter	Type	Description
	TCP Server Deux Serial	Text	TCP Server Deux serial

The method ***INIT_TCPsd*** initialises the TCP Server 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 calls to the BASh and TCP Deux components but before any other calls to the TCP Server Deux component package.

TCP Deux Server Serial is the TCP Server Deux serial which came with your purchase of the TCP Server Deux component package. A single TCP Server Deux serial provides for use of the TCP Server Deux component package on all platforms. If the TCP Server Deux package is being tested or being used in demonstration mode, use an empty serial number; this will allow 30 minutes of unlimited use.

TCPsd_Clear_Services_All_s

TCPsd_Clear_Services_All_s

TCPsd_Clear_Services_All_s

	Parameter	Type	Description

The method ***TCPsd_Clear_Services_All_s*** will clear the all services from the **TCPsd Services Stack**. The TCPsd server can not be running when this method is called.

Note: The TCPsd server can not be running when this method is called. If the TCP Server is running, call the method ***TCPsd_Stop_Server*** to stop the TCPsd server before clearing services.

TCPsd_Clear_Service_s

TCPsd_Clear_Service_s *(Service Name)*

TCPsd_Clear_Service_s

(

-> *Service Name :* String[32]

)

	Parameter	Type	Description
	Service Name	String[32]	Name of Service to be cleared

The method ***TCPsd_Clear_Service_s*** will clear the specified service from the **TCPsd Services Stack**. The TCPsd server can not be running when this method is called.

Note: The TCPsd server can not be running when this method is called. If the TCP Server is running, call the method ***TCPsd_Stop_Server*** to stop this server before clearing the service.

Service Name is the unique service name assigned when creating the service with ***TCPsd_Create_Service_s***.

TCPsd_Create_Service_s

TCPsd_Create_Service_s *(Service Name ; Local IP Address ; Local Port ; Remote Port ; Number of Listeners ; Send Timeout ; Protocol ; Remote IP Address ; Before Handler Method Name ; During Handler Method Name ; After Handler Method Name ; Post Processor Handler Method Name ; SSL Certificate Full Path ; SSL Private Key Full Path ; SSL Private Key Password) => qi Service Created*

TCPsd_Create_Service_s

(

-> *Service Name :* String[32]
-> *Local IP Address :* Longint
-> *Local Port :* Longint
-> *Remote Port :* Longint
-> *Number of Listeners :* Longint
-> *Send Timeout :* Longint
-> *Protocol :* Longint
-> *Remote IP Address :* Longint
-> *Before Handler Method Name :* String[32]
-> *During Handler Method Name :* String[32]
-> *After Handler Method Name :* String[32]
-> *Post Processor Handler Method Name :* String[32]
-> *SSL Private Key Full Path :* Text
-> *SSL Certificate Full Path :* Text
-> *SSL Private Key Password :* Text

)

=> *qi Service Created :* Longint

	Parameter	Type	Description
	Service Name	String[32]	Unique name for new TCPsd Service
	Local IP Address	Longint	IP address of local host to accept connections on
	Local Port	Longint	Port of local host to accept connections on
	Remote Port	Longint	Port of remote host to accept connections from
	Number of Listeners	Longint	Number of TCP listeners to open for this service
	Send Timeout	Longint	Send timeout to be used on all TCP listeners for this service
	Protocol	Longint	TCPd Protocol to be used for this service's listeners
	Remote IP Address	Longint	IP address of remote host to accept connections from

	Parameter	Type	Description
	Before Handler Method Name	String[32]	Method to be executed in the Before phase of an established connection
	During Handler Method Name	String[32]	Method to be executed in the During phase of an established connection
	After Handler Method Name	String[32]	Method to be executed in the After phase of an established connection
	Post Processor Handler Method Name	String[32]	Method to be executed after a close has been issued for an established connection
	SSL Private Key Full Path	Text	Full path to SSL certificate file (ITK v2.5.x only)
	SSL Certificate Full Path	Text	Full path to SSL private key file (ITK v2.5.x only)
	SSL Private Key Password	Text	Password for SSL private key (ITK v2.5.x only)
	qi Service Created	Longint	qi for Service successfully created in TCPsd Services Stack

The method ***TCPsd_Create_Service_s*** creates a new service in the **TCPsd Services Stack**. This does not actually start the server; you'll need to call the method ***TCPsd_Start_Server*** for your server to start responding to requests.

Service Name is the unique, non-empty service name used to identify this service in the TCPsd Services Stack. You will need to use this name to clear this service from the stack as well.

Local IP Address is the local IP address for this service to use. This is useful when the server is configured with multiple IP addresses. Set *Local IP Address* to zero (0) to allow this service to listen on all of the IP addresses assigned to the server. Setting this parameter to MAXLONG will allow this service to listen on only the primary IP address on the local machine.

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

Remote Port is the remote port to accept connections from. Set *Remote Port* to zero (0) to accept connections from any remote port.

Number of Listeners is the number of TCP listening streams to be opened for this service. These streams will be managed by TCP Server Deux.

Send Timeout is the default send timeout in seconds for all TCP sending on the TCP communications streams for this service. 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.

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 (documented separately in this manual).

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

Before Handler Method Name is the name of the method to be executed in the Before phase of a connection. Typically, this method would get and parse the request information. Passing an empty value for this parameter is allowed.

During Handler Method Name is the name of the method to be executed in the During phase of a connection. Typically, this method would process the request and build the information to be sent back to the requestor. Passing an empty value for this parameter is allowed.

After Handler Method Name is the name of the method to be executed in the After phase of a connection. Typically, this method would send the data built in the during phase back to the requestor. Passing an empty value for this parameter is allowed.

Post Processor Handler Method Name is the name of the method to be executed after the connection has been com-

pletely handled and is being closed. Typically, this method would be used for handling maintenance and logging functionality within a server. Passing an empty value for this parameter is allowed.

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

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

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

qi Service Created is the indicator for whether this service has been created in the **TCPsd Services Stack**. *qi Service Created* will be set to zero (0) if the service is not created and will be set to one (1) if the service is created successfully.

TCPsd_ERROR

TCPsd_ERROR *(TCPsd Error Number; Special Error Text; Calling Method Name)*

TCPsd_ERROR

(

-> *TCPsd Error Number :* Longint
-> *Special Error Text :* Text
-> *Calling Method Name :* Text

)

	Parameter	Type	Description
	TCPsd Error Number	Longint	Internal TCPsd 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 ***TCPsd_ERROR*** acts as a callback method from within the TCPsd module for errors that may occur. Any time an error condition is detected within the TCPsd module, a call to the method ***TCPsd_ERROR*** is made.

The internal *TCPsd 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 Server Deux method which call the ***TCPsd_ERROR*** method.

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

TCPsd_Get_TimeLog_FullPath

TCPsd_Get_TimeLog_FullPath => *Full Path*

TCPsd_Get_TimeLog_FullPath
=> *Full Path : Text*

	Parameter	Type	Description
	Full Path	Text	Full path to time log document

The method ***TCPsd_Get_TimeLog_FullPath*** returns the full path to the **Time Log** document for the current applica-

tion. The Time Log document does not necessarily exist nor does the Time Log functionality need to even be enabled. Rather, this method returns the full path of where the Time Log document would be located if the Time Log was enabled and the document already existed.

Full Path is the full path to the Time Log document for the current 4D structure.

Note: this method was added to TCP Server Deux v1.0.0b02.

TCPsd_qi_Server_Running

TCPsd_qi_Server_Running *=> qi Server Running*

TCPsd_qi_Server_Running
=> qi Server Running : Longint

	Parameter	Type	Description
	qi Server Running	Longint	qi for whether the TCPsd Server is currently running

The method ***TCPsd_qi_Server_Running*** returns an indicator for whether the TCPsd Server is currently running. When the TCPsd Server is running, the services set up using ***TCPsd_Create_Service_s*** have listeners open waiting for connections.

qi Server Running is the indicator for whether the TCPsd Server is currently running. *qi Server Running* will be set to zero (0) if the server is not running and will be set to one (1) if the server is running.

TCPsd_qi_TimeLog_Running

TCPsd_qi_TimeLog_Running *=> qi Time Log Running*

TCPsd_qi_TimeLog_Running
=> qi Time Log Running : Longint

	Parameter	Type	Description
	qi Time Log Running	Longint	qi for whether the Time Log is currently enabled

The method ***TCPsd_qi_TimeLog_Running*** returns an indicator for whether the **Time Log** is currently enabled. When the Time Log is enabled, connections made to the TCP Server Deux services will generate Time Log entries to the Time Log document.

qi Time Log Running is the indicator for whether the Time Log is currently enabled. *qi Time Log Running* will be set to zero (0) if the Time Log is not enabled and will be set to one (1) if the Time Log is enabled.

Note: this method was added to TCP Server Deux v1.0.0b02.

TCPsd_Set_PostProcessorMethod_s

TCPsd_Set_PostProcessorMethod_s *(Service Name ; Post Processor Method Name)*
=> qi Handler Set

TCPsd_Set_PostProcessorMethod_s
(
-> Service Name : String[32]
-> Post Processor Method Name : String[32]
)
=> qi Handler Set : Longint

	Parameter	Type	Description
	Service Name	String[32]	Full name of existing service to set the post processor handler method name for
	Post Processor Method Name	String[32]	Post processor method name to be set
	qi Handler Set	Longint	qi for whether method name was successfully set

The method ***TCPsd_Set_PostProcessorMethod_s*** will set the post processor method name for a specified TCPsd Service.

Note: The post processor method name can only be changed when the TCPsd Server is not running. To determine if the TCPsd Server is running, use the ***TCPsd_qi_Server_Running*** method. To start or stop the server, use the ***TCPsd_Start_Server*** or ***TCPsd_Stop_Server*** methods.

Service Name is the full name of the TCPsd Service which is going to have the post processor method name set. The TCPsd service named by *Service Name* must already exist in the TCPsd Services Stack.

Post Processor Method Name is the full method name to set as the post processor handler for the specified TCPsd Service.

qi Handler Set is the indicator for whether the TCPsd Service post processor handler was successfully set. *qi Handler Set* will be set to zero (0) if the handler was not changed and will be set to one (1) if the handler was changed successfully.

Note: this method was added to TCP Server Deux v1.0.0b02.

TCPsd_Set_Server_Prefs

TCPsd_Set_Server_Prefs *(Minimum Handlers ; Server Throttle ; qi Local Handler Processes) => qi Preferences Set*

TCPsd_Set_Server_Prefs
(
-> *Minimum Handlers :* Longint
-> *Server Throttle :* Longint
-> *qi Local Handler Processes :* Longint
)
=> *qi Preferences Set :* Longint

	Parameter	Type	Description
	Minimum Handlers	Longint	Minimum number of handler processes to create when starting the server
	Server Throttle	Longint	Throttle Setting for server processing
	qi Local Handler Processes	Longint	qi for whether handler process are local processes
	qi Preferences Set	Longint	qi for whether settings were changed successfully

The method ***TCPsd_Set_Server_Prefs*** will set preferences used by the TCPsd Server. These preferences are global, and affect the operation of the TCPsd Server. They are not specific to any individual service.

Note: The preferences can only be changed when the TCPsd Server is not running. To determine if the TCPsd Server is running, use the ***TCPsd_qi_Server_Running*** method. To start or stop the server, use the ***TCPsd_Start_Server*** or ***TCPsd_Stop_Server*** methods.

Minimum Handlers is the minimum number of handler processes to create when the TCPsd Server is started. These processes can be either local or global processes.

Server Throttle is the throttle setting for TCPsd Server. Valid values are 0 to 99, with higher values indicating less processing time being given to the TCPsd Server. Use –1 to maintain the current setting. When less processing time is being given to the TCPsd Server, the server will be slower to respond to requests.

qi Local Handler Processes is qi for whether the handler process created by the TCPsd Server are local processes. Pass 1 for the handlers to be local processes, and pass 0 for the handlers to be global processes. Remember, local processes cannot access the database when running in client/server mode.

qi Preferences Set is the indicator for whether the TCPsd Server preferences were successfully changed. *qi Preferences Set* will be set to zero (0) if the preferences were not

changed and will be set to one (1) if the preferences are changed successfully.

TCPsd_Start_Server

TCPsd_Start_Server

TCPsd_Start_Server

	Parameter	Type	Description

The method ***TCPsd_Start_Server*** starts the TCPsd Server. The handler processes are spawned, and the TCP listeners specified by the services created with ***TCPsd_Create_Service_s*** are opened. To stop the TCPsd Server, call the method ***TCPsd_Stop_Server***. If there are currently no services in the **TCPsd Services Stack** then this method will spawn an error and the TCP server will not be started.

TCPsd_Start_TimeLog

TCPsd_Start_TimeLog

TCPsd_Start_TimeLog

	Parameter	Type	Description

The method ***TCPsd_Start_TimeLog*** starts the TCPsd Time Log. See the section **"Time Log" on page 21**, above, for details on the TCPsd Time Log.

Note: this method was added to TCP Server Deux v1.0.0b02.

TCPsd_Stop_Server

TCPsd_Stop_Server

TCPsd_Stop_Server

	Parameter	Type	Description

The method ***TCPsd_Stop_Server*** stops the TCPsd Server. All listeners managed by TCPsd Server are stopped, and the handler processes will finish and die. The TCPsd Server will no longer listen for or respond to TCP connections. To start the TCPsd Server, call the method ***TCPsd_Start_Server***.

TCPsd_Stop_TimeLog

TCPsd_Stop_TimeLog

TCPsd_Stop_TimeLog

	Parameter	Type	Description

The method ***TCPsd_Stop_TimeLog*** stops the TCPsd Time Log. See the section **"Time Log" on page 21**, above, for details on the TCPsd Time Log.

Note: this method was added to TCP Server Deux v1.0.0b02.

Version History

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

TCP Server Deux v1.0.0

released 20020322

Changes:

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

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.

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

Increased minimum version of TCPd compatible with this release to v1.1.0.

TCP Server Deux v1.0.0b02

released 20020109

Changes:

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

Added serialization check before calling any callback methods within component.

Added post processor handler method name to TCP Services stack; implemented post processor handler method within connection handler controller.

Added post processor handler method name parameter to the method ***TCPsd_Create_Service_s***.

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

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

Added protected methods:

TCPsd_Get_TimeLog_FullPath
TCPsd_qi_TimeLog_Running
TCPsd_Set_PostProcessorMethod_s
TCPsd_Start_TimeLog
TCPsd_Stop_TimeLog

TCP Server Deux v1.0.0b01

released 20011008

Changes:

First public beta release of the component.

Errors

The listing of error codes and conditions is obviously a continuously updated process. With practically every new version of TCP Server 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 Server Deux.

Different methods in the TCP Server 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 Server 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 Server 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 Server Deux component.

Error Codes

When an "_ERROR" method is called in the TCP Server 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.

TCPsd: 29007
(gap)
02 Failed to create TCPsd handler process.
03 Component not serialized.
04 Version of BASh being used is too old.
05 Long hard name for compatible BASh not correct.
06 Version of TCP Deux being used is too old.
07 Long hard name for compatible TCP Deux not correct.
08 Service name is empty.
09 No services created to launch server with.
(gap)
15 Failed to create TCPsd time log document.
16 Failed to failed TCPsd time log document for writing.
17 TCPsd Time Log Stack index is negative.
18 TCPsd Time Log Stack index is out of range.

Method Listing

The following is a listing of all of the methods within the TCP Server 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 Server Deux component is installed. Following each method is a list of the error codes which each method can generate.

ENV_Get_TCPsd_HardName_Long
ENV_Get_TCPsd_HardName_Short
INIT_TCPsd
TCPsd_Check_for_Requests
TCPsd_Check_Serial_Full
TCPsd_Check_Serial_Quick
TCPsd_Clear_Services_All_s
TCPsd_Clear_Service_s
TCPsd_Create_Handler_Process
    2900702
TCPsd_Create_Service_s
    2900708
TCPsd_ERROR
TCPsd_Finish_TimeLog_Entry_s
    2900717
    2900718
TCPsd_Get_AfterMeth_by_Index_s
TCPsd_Get_BeforeMeth_by_Index_s
TCPsd_Get_DuringMeth_by_Index_s
TCPsd_Get_FreeHandler_ProcessID
TCPsd_Get_FreeIndex_s
TCPsd_Get_HandlerProcess_Count
TCPsd_Get_HandlerProsName_Root
TCPsd_Get_Index_by_StrmRef_s
TCPsd_Get_Index_s
TCPsd_Get_IPAddrs_by_LPort_s
TCPsd_Get_PostMeth_by_Index_s
TCPsd_Get_ServiceNames_All_s
TCPsd_Get_TimeLog_FullPath
TCPsd_Handler_c
TCPsd_INIT
    2900703
TCPsd_Launch_Listeners
TCPsd_Lock_Stack_s
    2902015
TCPsd_Manager_c
TCPsd_qi_BASh_Version
    2900704
    2900705
TCPsd_qi_IPPort_Free_s
TCPsd_qi_Server_Running
TCPsd_qi_TCPd_Version
    2900706
    2900707
TCPsd_qi_TimeLog_Running
TCPsd_Send_Serial_Info
TCPsd_Set_PostProcessorMethod_s
    2900708

TCPsd_Set_Server_Prefs
TCPsd_Set_StackSize_s
TCPsd_Set_TimeLog_Entry_s
	2900717
TCPsd_Start_Server
	2900702
	2900703
	2900709
TCPsd_Start_TimeLog
	2900715
	2900716
TCPsd_Stop_Server
TCPsd_Stop_TimeLog
TCPsd_Unlock_Stack_s