HTTP Server Deux

Developer Documentation
v1.0.0b02

Deep Sky Technologies, Inc.
P.O. Box 6897
Vero Beach, FL 32961-6897
561.794.9494

http://www.deepskytech.com/

Software Engineers
James A. Crate
Robert T. McGoye
Steven G. Willis

Manual
James A. Crate
Steven G. Willis

Software License and Limited Warranty

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

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

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

> **a.** You may make backup copies of the Software for your use provided they bear the DSTi copyright notice.

> **b.** You may use this Software in an unlimited number of distributed copies of a single application or database. Use in additional applications requires separate licenses for the use of this Software.

> **c.** Distribution or dissemination of the software license serial is strictly prohibited. This license grants the original licensee sole use of the license serial for enabling this Software.

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

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

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

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

6. Warranty Disclaimer: The software is provided "as is" without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. DSTi does not warrant, guarantee, or make any representations regarding the use, or the results of the use, of the software or written materials in the terms of correctness, accuracy, reliability, currentness or otherwise. The entire risk as to the results and performance of the software is assumed by the Licensee. If the software or written materials are defective you, and not DSTi or 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

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

BASh, TCP Deux, SMTP Deux, POP3 Deux, FTP Client Deux, HTTP Client Deux, eTrans, TCP Server Deux, HTTP Server Deux, HTTP Log Deux are copyright Deep Sky Technologies, Inc.

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

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

Macintosh and MacOS are trademarks of Apple Computer, Inc.

Windows is a trademark of Microsoft Corporation.

Table of Contents

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

Index

Preface

The HTTP Server Deux component is designed to work in conjunction with many other components. Specifically, the HTTP Server Deux component requires that the TCP Server Deux and TCP Deux components already be installed in your database structure file. Additionally, these components require that BASh, available for free from DSTi, also be installed already in your database structure file. Details and documentation for these components are provided separately.

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.

Acknowledgements

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

In particular, the efforts of Robert T. McGoye have contributed much to the HTTP 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 HTTP Server Deux component.

Later tweaks and additions to the HTTP 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 Deux components.

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

Features

HTTP Server Deux is a 4th Dimension component intended to work with TCP Server Deux to provide the functionality necessary to implement a web server. With HTTP Server Deux, it is possible to set up a web server in a 4D database in a matter of minutes!

With the HTTP Server Deux component, a 4th Dimension developer can concentrate on the code necessary to build HTML responses to requests. The developer will not have to worry about how to properly parse the HTTP request or build a proper response header. However, since all data is available to the developer, any custom parsing desired could be performed, and there is also some flexibility in setting the header values.

System Requirements

The HTTP 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.

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 HTTP Server 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:

<div align="center">http://www.deepskytech.com/</div>

If there are terms or conventions which you find difficult to understand in relation to the HTTP 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 and Updating HTTP Server Deux

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

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

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

4D Insider parses the HTTP 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 HTTP Server Deux component.

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

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

Save the database in 4D Insider.

Place a copy of the Affix HTTPsd document in the 4DX folder.

The Affix HTTPsd document contains essential data and resources for many of the methods within the HTTP Server Deux component. For many of the methods within the HTTP Server Deux component to function properly, the Affix HTTPsd document must be in the 4DX folder for the current structure.

On Macintosh, the Affix HTTPsd document is entitled **Affix_HTTPs_Deux_MacOS9.4DX** and is located in the Mac4DX directory in the HTTP Server Deux component's archive. The document should be copied into the Mac4DX folder of your current structure. If the HTTP Server Deux component is going to be used in all of your 4D projects, the Affix HTTPsd document can instead be placed within the Mac4DX folder within the 4D folder of your system.

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

be used in all of your 4D projects, the Affix HTTPsd 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 HTTPsd document should be installed.

Call the method *INIT_HTTPsd* **early in the On Startup database method.**

To initialize the TCP Server Deux component in your code, place a call to the method *INIT_HTTPsd* early in your **On Startup** database method. However, this method must be `after` the calls to *INIT_BASh*, *INIT_TCPd,* and *INIT_TCPsd*.

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

The HTTP 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 HTTP 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 HTTP 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 HTTP 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 HTTP Server Deux installation process, alerts you through a dialog box and saves the list of objects in conflict. This list is stored as a text file in the 4D database folder.

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

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

Affix HTTPsd Document

The Affix HTTPsd document (entitled **Affix_HTTPs_Deux_MacOS9.4DX** on Macintosh; entitled **Affix_HTTPs_Deux.4DX** and **Affix_HTTPs_Deux.RSR** on Windows) contains essential data and resources for many of the methods within the HTTP Server Deux component. For many of the methods within the HTTP Server Deux component to function

properly, the Affix HTTPsd document must be in the 4DX folder for the current structure. For distributed and installed versions of your 4D projects, the Affix HTTPsd document must be available in the 4DX folder for the HTTP Server Deux methods to continue to function properly.

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

Uninstalling HTTP Server Deux

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

To uninstall HTTP Server Deux from your 4D database:

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

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

Consider again how great the HTTP 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_HTTPsd* from your On Startup **database method.**

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

HTTP Server Deux Conventions

Throughout this manual, and all other documentation and supporting materials, included with the HTTP 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 HTTP 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 HTTP Server Deux component or TCP networks in general, feel free to contact Deep Sky Technologies, Inc., support. We will be more than happy to help you in any way we reasonably can. And, only through your questions do we know what subjects to include in future versions of this manual.

HTTPsd Services

HTTP Server Deux uses a concept called **services**. A **service** will receive connections on 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 you've read the TCP Server Deux documentation, you may be thinking that this sounds familiar. You would be correct. TCP Server Deux uses services too, in the same way. In fact, when you configure an HTTP Server Deux service, HTTP Server Deux configures a TCP Server Deux service. So, you only need to declare a service once. If you attempt to declare your service through both TCP Server Deux and HTTP Server Deux, it won't work.

Using a service is very simple. When you configure a service, you specify a handler method name. This handler method will be executed when an HTTP request is received. In this handler method, you can access the fields of the HTTP request, and build a reply to send back to the web browser. Really, that's all there is to it!

HTTPsd Services Stack

The HTTPsd Services Stack is an internal mechanism within the HTTP Server Deux component to efficiently keep track of the active servers. This allows you to set up multiple servers on different IP addresses or ports. For instance, you could set up a production HTTP server on port 80 on one IP address, and an administrative HTTP interface on port 80 on a different IP address, all handled within the same 4D application.

As mentioned before, the HTTPsd Services Stack interfaces directly with the TCPsd Services Stack. In order to use HTTP Server Deux, you must set up HTTPsd Services.

The HTTPsd 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 HTTPsd Services Stack is basically just a listing of data about each HTTP server managed by the HTTP Server Deux component. For each HTTP server managed by the HTTP Server Deux component, there is one row in the HTTPsd Services Stack; uniqueness within the HTTPsd 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 HTTPsd 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
Handler Method Name	String [32]

The service name is a unique name used to reference each HTTPsd Service.

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, to use 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 HTTP Request

HTTP Server Deux parses the complete HTTP request for the developer. The request is broken up into header elements and body elements. The request header contains all information about the browser, what is being requested, and what can be accepted. The request body contains extra information the developer will want to use to handle the request, such as extra fields, or information the user entered into a form on the web page.

The HTTP request and response are both defined in the HTTP RFC (#1945). For convenience, you can access this RFC at:

> http://www.deepskytech.com/rfcs/rfc_1945.txt

The HTTP Request Header

The HTTP Request header is parsed out into the individual selectors. HTTP Server Deux provides convenience methods to access the titles and values of these selectors. Constants are provided to make it easy and consistent to specify which selector is desired. Following is a list of all the HTTP Server Deux Request selectors available in the **HTTPq Selectors** custom constant group:

Constant	Description
HTTPq_ACEN	Accept-Encoding
HTTPq_ACHT	Accept-Charset
HTTPq_ACLG	Accept-Language
HTTPq_ACPT	Accept Selector
HTTPq_addr	Client Address DNS
HTTPq_Agnt	User Agent
HTTPq_AUTH	User Authenticate
HTTPq_CNTN	Connection Selector
HTTPq_COOK	Cookie Selector
HTTPq_CTBD	Content Type Boundary
HTTPq_CTLN	Content Length
HTTPq_ctyp	Content Type
HTTPq_DIRE	FSSpec pointing to the root folder of the virtual host
HTTPq_Direct	Direct argument - follows the $ in a URL
HTTPq_FILE	File requested
HTTPq_FPTH	Path to file requested
HTTPq_HOST	Host
HTTPq_Kact	Action being performed (cgi, acgi, preprocessor, etc.)
HTTPq_Kapt	Action path
HTTPq_Kcid	A unique longint identifying the TCP/IP connection
HTTPq_Kcip	Client IP address
HTTPq_kfor	Search arguments – follows the ? in a URL
HTTPq_meth	HTTP method (GET, POST, etc.)
HTTPq_pass	Password
HTTPq_refr	Referring URL
HTTPq_RURL	Relative URL
HTTPq_scnm	Path of the CGI being executed
HTTPq_svnm	Domain name of the server
HTTPq_svpt	Port the server is listening on
HTTPq_user	Username
HTTPq_MISC	Miscellaneous

The HTTP Request Body

The HTTP Request body is parsed out into the individual fields. These fields, their names and values, are stored in the HTTPq Request Fields Stack. This stack also stores information specifying whether each field was in the GET or POST portion of the request body, since it is possible to use both in the same request, and the developer may want to know that information.

Convenience methods are provided to access the fields names and values, or determine the number of fields in the request. If you work with forms with scrolling lists, you'll appreciate the features that let you determine how many fields have the same name, and access the Nth occurrence of those fields.

Constants are provided to aid the developer when accessing fields, for specifying the request type. These constants are in the **HTTPq Request Types**custom constants group. A listing of these constants follows:

Constant	Description
HTTPq_GET_Type	Access fields only from the GET portion of the request
HTTPq_POST_Type	Access fields only from the POST portion of the request
HTTPq_Both_Type	Access fields in either the GET or POST portions

The HTTP Response

HTTP Server Deux makes it easy for the developer to build a response. The response header information is stored in a selector stack. Before the response body is sent, the response header is built from these selectors and sent. A convenience method is provided that sets all selectors to default values for the developer.

The developer can easily modify the response header, by resetting the values of the desired selectors. Convenience methods are provided to get and set the values of the selectors. Constants are provided to make it easy and consistent to specify which selector should be accessed. Following is a list of all the HTTP Server Deux Response selectors available in the **HTTPp Selectors**custom constant group:

Constant	Description
HTTPp_allw	Allow
HTTPp_auth	WWW-Authenticate
HTTPp_cnct	Connection
HTTPp_COOK	Cookie selector
HTTPp_cten	Content-Encoding
HTTPp_ctlg	Content-Language
HTTPp_ctln	Content-Length
HTTPp_ctyp	Content-Type
HTTPp_date	Date
HTTPp_expr	Expires
HTTPp_link	Link
HTTPp_lmod	Last-Modified
HTTPp_loct	Location
HTTPp_mimv	Mime-Version
HTTPp_prag	Pragma
HTTPp_rcod	Response Code
HTTPp_rtra	Retry-After
HTTPp_srvr	Server
HTTPp_optl	Optional response text
HTTPp_URI	URI
HTTPp_MISC	Miscellaneous

HTTP Response Codes

The HTTP protocol has many response codes. These codes indicate to the web browser the condition of the response. These codes are defined in the HTTP RFC (#1945). Section 9 of this RFC contains detailed explanations of each HTTP response code presented here. Here is a basic breakdown of the code scheme:

- 1xx: Informational – Reserved for future use

- 2xx: Success – The action was successfully received, understood, and accepted

- 3xx: Redirection – Further action must be taken to complete the request

- 4xx: Client Error – The request contains bad syntax or cannot be fulfilled

- 5xx: Server Error – The server failed to fulfill an apparently valid request

The HTTP Server Deux component provides constants to allow you to set the response code. These constants are detailed below:

Constant	HTTP Code	Description
HTTP_100	100	Continue
HTTP_101	101	Switching Protocols
HTTP_200	200	OK
HTTP_201	201	Created
HTTP_202	202	Accepted
HTTP_203	203	Non-Authoritative Information
HTTP_204	204	No Content
HTTP_205	205	Reset Content
HTTP_206	206	Partial Content
HTTP_300	300	Multiple Choices
HTTP_301	301	Moved Permanently
HTTP_302	302	Moved Temporarily
HTTP_303	303	See Other
HTTP_304	304	Not Modified
HTTP_305	305	Use Proxy
HTTP_400	400	Bad Request
HTTP_401	401	Unauthorized
HTTP_402	402	Payment Required
HTTP_403	403	Forbidden
HTTP_404	404	Not Found
HTTP_405	405	Method Not Allowed
HTTP_406	406	Not Acceptable
HTTP_407	407	Proxy Authentication Required
HTTP_408	408	Request Timeout
HTTP_409	409	Conflict
HTTP_410	410	Gone
HTTP_411	411	Length Required

HTTP_412	412	Precondition Failed
HTTP_413	413	Request Entity Too Large
HTTP_414	414	URI Too Large
HTTP_415	415	Unsupported Media Type
HTTP_500	500	Internal Server Error
HTTP_501	501	Not Implemented
HTTP_502	502	Bad Gateway
HTTP_503	503	Service Unavailable
HTTP_504	504	Gateway Timeout
HTTP_505	505	HTTP Version Not Supported

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 valid value format.

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

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

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

TCPd Protocols

The TCPd_Protocols, available within the TCP Deux component, 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_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.

HTTP Server Deux Adjunct Components

HTTP Server Deux has been expanded to allow the use of other components for additional functionality. Adjunct components can include code that runs when a request is handled. This allows for other components to handle additional functionality desired in a web server such as logging requests to a file.

The adjunct components are specified in the constant group HTTPsd Adjunct Component IDs. These adjunct components can be registered, and HTTP Server Deux will allow them to function.

The following is a listing the constants, and their values, within the HTTPsd Adjunct Component IDs constant group:

Constant	Value
HTTPsd_AdjC_Logger	1

Modules

All of the code within the HTTP 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 HTTP Server Deux component.

Each module contains a set of methods which can be used to easily implement a web server once the HTTP 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.

General Methods

These methods are not tied to a specific aspect of the HTTP Server Deux functionality. They perform general or utilitarian functions.

ENV_Get_HTTPsd_HardName_Long

ENV_Get_HTTPsd_HardName_Long => *Long Hard Name*

ENV_Get_HTTPsd_HardName_Long
        => *Long Hard Name* : **Text**

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

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

Long Hard Name is the full, hard coded name of the HTTP Server Deux component. As of this release, this will always return the value "HTTP_Server_Deux_v1.0.0b02".

INIT_HTTPsd

INIT_HTTPsd (*HTTP Server Deux Serial*)

INIT_HTTPsd
    (
        -> *HTTP Server Deux Serial* : **Text**
)

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

The method *INIT_HTTPsd* initializes the HTTP 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 components BASh, TCP Deux, and TCP Server Deux, but before any other calls to the HTTP Server Deux component package.

HTTP Deux Server Serial is the HTTP Server Deux serial which came with your purchase of the HTTP Server Deux component package. A single HTTP Server Deux serial provides for use of the HTTP Server Deux component package on all platforms. If the HTTP 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.

HTTPp Module

The HTTPp Module is for setting up the HTTP response to be sent back to the requestor. Methods are provided to allow you to work with the response header selectors, as well as set the response content. Convenience methods are provided to set up a default response header, so you don't have to configure the response header if you don't want to do so.

_____

HTTPp_Append_Reply_x

HTTPp_Append_Reply_x (*Text to Append*) => *Bytes in Buffer*

HTTPp_Append_Reply_x
   (
       -> *Text to Append* : **Text**
)
       => *Bytes in Buffer* : **Longint**

Parameter	Type	Description
Text to Append	**Text**	**Text to be appended to HTTP Server Deux reply buffer**
Bytes in Buffer	**Longint**	**Number of bytes in HTTP Server Deux reply buffer, or error code if less than 0**

The method *HTTPp_Append_Reply_x* will append the text to the HTTP Server Deux reply buffer. The HTTP Server Deux reply buffer is discrete for each HTTP Server Deux handler. When the HTTPsd Service handler is finished, the reply buffer is flushed to the web browser.

Text to Append is the text to be appended to the HTTP Server Deux reply buffer. Normally, this will be HTML for the response to the request currently being handled.

Bytes in Buffer is the number of bytes currently in the HTTP Server Deux reply buffer. If *Bytes in Buffer* is less than 0, it is an error code.

HTTPp_Append_Reply_z

HTTPp_Append_Reply_z (*BLOB to Append*) => *Bytes in Buffer*

HTTPp_Append_Reply_z
    (
        -> *BLOB to Append* : **Pointer**
)
        => *Bytes in Buffer* : **Longint**

Parameter	Type	Description
BLOB to Append	**Pointer**	**Reference to BLOB to be appended to HTTP Server Deux reply buffer**
Bytes in Buffer	**Longint**	**Number of bytes in HTTP Server Deux reply buffer, or error code if less than 0**

The method *HTTPp_Append_Reply_z* will append the BLOB to the HTTP Server Deux reply buffer. The HTTP Server Deux reply buffer is discrete for each HTTP Server Deux handler. When the HTTPsd Service handler is finished, the reply buffer is flushed to the web browser.

BLOB to Append is a reference to the BLOB to be appended to the HTTP Server Deux reply buffer. Normally, this will be HTML for the response to the request currently being handled.

Bytes in Buffer is the number of bytes currently in the HTTP Server Deux reply buffer. If *Bytes in Buffer* is less than 0, it is an error code.

HTTPp_ERROR

HTTPp_ERROR (*HTTPp Error Number* ; *Special Error Text* ; *Calling Method Name*)

HTTPp_ERROR
    (
        -> *HTTPp Error Number*: **Longint**

-> *Special Error Text*: **Text**
-> *Calling Method Name*: **Text**
)

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

The internal *HTTPp 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 HTTPp method which called the **HTTPp_ERROR** method.

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

_____

HTTPp_Get_ResponseCode

HTTPp_Get_ResponseCode => *Response Code*

HTTPp_Get_ResponseCode
=> *Response Code* : **Longint**

Parameter	Type	Description
Response Code	Longint	Current value of the HTTPp Response Code selector

The method *HTTPp_Get_ResponseCode* returns the current value of the HTTPp Response Code selector.

Response Code is the currently set HTTPp Response Code selector value for the current handler.

Note: this method was added with HTTP Server Deux v1.0.0b02.

_____

🔳 HTTPp_Get_Selector_Size_s

HTTPp_Get_Selector_Size_s => *Number of Selectors*

HTTPp_Get_Selector_Size_s
>> => *Number of Selectors* : **Longint**

Parameter	Type	Description
Number of Selectors	**Longint**	**The size of the response selector stack**

The method *HTTPp_Get_Selector_Size_s* returns the current size of the HTTPp response selectors stack. This would allow you to loop through the HTTPp response selector stack to build a custom response if you needed to do so.

Number of Selectors is the size of the response selector stack.

_____

🔳 HTTPp_Get_Selector_Title_s

HTTPp_Get_Selector_Title_s (*HTTPp Selector Constant*) => *Selector Title*

HTTPp_Get_Selector_Title_s
>> (
>>>> -> HTTPp *Selector Constant* : **Longint**
>>)
>>>> => *Selector Title* : **Text**

Parameter	Type	Description
HTTPp Selector Constant	**Longint**	**Index into HTTPp Selector Stack**
Selector Title	**Text**	**Title of specified selector**

The method **HTTPp_Get_Selector_Title_s** will get the title of the specified HTTP response header selector. This can be useful if you want to build a custom HTTP response header.

HTTPp Selector Constant is the HTTPp Selector constant. This could also be an index used in a loop.

Selector Title is the current title of the response selector. The selector titles are constant, except for the HTTPp_rcod selector title, which depends on the selector value.

_____

HTTPp_Get_Selector_x_s

HTTPp_Get_Selector_x_s (*HTTPp Selector Constant* ; *Selector Title* ; *Max Length*) => *Response Selector Value*

HTTPp_Get_Selector_x_s
    (
        -> *HTTPp Selector Constant* : **Longint**
        -> *Selector Title* : **Text**
        -> *Max Length* : **Longint**
)
        => *Response Selector Value* : **Text**

Parameter	Type	Description
HTTPp Selector Constant	**Longint**	**Index into HTTPp Selector Stack**
Selector Title	**Text**	**Title of specified selector; only use this if response selector is HTTPp_MISC**
Max Length	**Longint**	**Max length of text to return**
Response Selector Value	**Text**	**Value of specified selector**

The method *HTTPp_Get_Selector_x_s* will get the value of the specified HTTP response header selector. This can be useful if you want to build a custom HTTP response header.

HTTPp Selector Constant is the HTTPp Selector constant. This could also be an index used in a loop.

Selector Title is the title of the response selector to access. You should only use this value if the *HTTPp Selector Constant* is HTTPp_MISC.

Max Length is the maximum text length of the selector value to return.

Response Selector Value is the value of the specified HTTP response header selector, returned as text.

_____

HTTPp_Get_Selector_z_s

HTTPp_Get_Selector_z_s (*HTTPp Selector Constant* ; *Selector Title* ; *Referenced BLOB*)

HTTPp_Get_Selector_z_s
   (
          -> *HTTPp Selector Constant* : **Longint**
          -> *Selector Title* : **Text**
          -> *Referenced BLOB* : **Pointer**
)

Parameter	Type	Description
HTTPp Selector Constant	**Longint**	**Index into HTTPp Selector Stack**
Selector Title	**Text**	**Title of specified selector; only use this if response selector is HTTPp_MISC**
Referenced BLOB	**Pointer**	**Reference to BLOB to contain specified response header selector value**

The method *HTTPp_Get_Selector_z_s* will put the value of the specified HTTP response header selector

into the referenced BLOB. This will not be limited to 32K like text variables.

HTTPp Selector Constant is the HTTPp Selector constant. This could also be an index used in a loop.

Selector Title is the title of the response selector to access. You should only use this value if the *HTTPp Selector Constant* is HTTPp_MISC.

Referenced BLOB is a reference to a BLOB to contain the value of the specified HTTPp response header selector.

_____

HTTPp_qi_Selector_Set_s

HTTPp_qi_Selector_Set_s (*HTTPp Selector Constant* ; *Selector Title*) => *qi Selector Set*

HTTPp_qi_Selector_Set_s
   (
        -> *HTTPp Selector Constant* : **Longint**
        -> *Selector Title* : **Text**
)
        => *qi Selector Set* : **Longint**

Parameter	Type	Description
HTTPp Selector Constant	**Longint**	**Index into HTTPp Selector Stack**
Selector Title	**Text**	**Title of specified selector; only use this if response selector is HTTPp_MISC**
qi Selector Set	**Longint**	**qi for whether specified HTTPp response header selector has been set**

The method **HTTPp_qi_Selector_Set_s** will let you determine if the specified HTTP response header selector value has been set.

HTTPp Selector Constant is the HTTPp Selector constant. This could also be an index used in a loop.

Selector Title is the title of the response selector to access. You should only use this value if the *HTTPp Selector Constant* is HTTPp_MISC.

qi Selector Set is qi for whether the specified HTTPp response header selector has been set. *qi Selector Set* will be set to zero (0) if the selector has not been set and will be set to one (1) if the selector has been set.

_____

HTTPp_Set_Cookie_Selector_s

HTTPp_Set_Cookie_Selector_s (*Cookie Name* ; *Cookie Value* ; *Cookie Domain* ; *Max Age* ; *Expiration Date* ; *Cookie Path* ; *qi Secure* ; *Cookie Comment*)

HTTPp_Set_Cookie_Selector_s
   (
         -> *Cookie Name* : **Text**
         -> *Cookie Value* : **Text**
         -> *Cookie Domain* : **Text**
         -> *Max Age* : **Longint**
         -> *Expiration Date* : **Text**
         -> *Cookie Path* : **Text**
         -> *qi Secure* : **Longint**
         -> *Cookie Comment* : **Text**
)

Parameter	Type	Description
Cookie Name	**Text**	**Name of cookie**
Cookie Value	**Text**	**Value of cookie**
Cookie Domain	**Text**	**Domain for which this cookie applies**
Max Age	**Longint**	**Max Age of cookie in seconds**
Expiration Date	**Text**	**Expiration Date for cookie; if *Max Age* is specified, this value will be ignored**
Cookie Path	**Text**	**Subset of URLs to which this cookie applies**
qi Secure	**Longint**	**qi for whether cookie is secure**
Cookie Comment	**Text**	**Comment for cookie**

The method **HTTPp_Set_Cookie_Selector_s** builds a cookie from the values specified and adds it to the

cookie selector value. No duplicate checking is performed, so calling this command multiple times with the same values will duplicate the cookies.

Cookie Name is the name for the cookie. Usually, a different name for each cookie is desirable. This value is required.

Cookie Value is value for the cookie. This value is required to build the cookie.

Cookie Domain is the domain for which this cookie applies. The web browser will send the cookie in the request header for any request to this domain. An explicitly specified domain must always start with a period. Pass a NULL value to skip this attribute.

Max Age is the maximum age of the cookie in seconds. After the specified number of seconds has passed, the web browser should discard the cookie. A value of zero (O) means the cookie should be discarded immediately. Pass a NULL value to skip this attribute.

Expiration Date is the expiration date for the cookie. After this date, the web browser should discard the cookie. The expiration date needs to be in the format "Weekday, DD-Mon-YY HH:MM:SS GMT". Pass a NULL value to skip this attribute.

Cookie Path is the subset of URLs to which this cookie applies. The web browser will only send the cookie if the requested path matches the path specified for the cookie. Pass a NULL value to skip this attribute.

qi Secure is qi for whether this cookie is secure. NOTE: the Secure attribute (with no value) directs the user agent to use only (unspecified) secure means to contact the origin server whenever it sends back this cookie. The user agent (possibly under the user's control) may determine what level of security it considers appropriate for "secure" cookies. The Secure attribute should be considered security advice from the server to the user agent, indicating that it is in the session's

interest to protect the cookie contents. Pass zero (0) to skip this attribute.

Cookie Comment is a comment to store with the cookie. NOTE: because cookies can contain private information about a user, the Cookie attribute allows an origin server to document its intended use of a cookie. The user can inspect the information to decide whether to initiate or continue a session with this cookie. Pass a NULL value to skip this attribute.

_____

HTTPp_Set_ResponseCode

HTTPp_Set_ResponseCode (*HTTP Response Code* ; *HTTP Response Text*)

HTTPp_Set_ResponseCode
   (
         -> *HTTP Response Code* : **Longint**
         -> *HTTP Response Text* : **Text (optional)**
)

Parameter	Type	Description
HTTP Response Code	**Longint**	**Index into HTTPp Selector Stack**
Selector Title	**Text**	**Optional HTTP Response Text**

The method **HTTPp_Set_ResponseCode** sets the response code for the response in the HTTP response header selector stack.

HTTP Response Code is the response code for the HTTP response. This value is set in the HTTPp_rcod selector. HTTP Response Code custom constants can be used in this parameter.

HTTP Response Text is an optional parameter that can specify the optional response text. If this parameter is specified, its value is set in the HTTPp_optl selector. Use this value to specify the realm for HTTP 401 authentication, or the URL for redirection on an HTTP 301 response.

HTTPp_Set_Selector_x_s

HTTPp_Set_Selector_x_s (*HTTPp Selector Constant* ; *Selector Title* ; *Selector Value* ; *qi Append Value*)

HTTPp_Set_Selector_x_s
    (
            -> *HTTPp Selector Constant* : **Longint**
            -> *Selector Title* : **Text**
            -> *Selector Value* : **Text**
            -> *qi Append Value* : **Longint**
)

Parameter	Type	Description
HTTPp Selector Constant	**Longint**	**Index into HTTPp Selector Stack**
Selector Title	**Text**	**Title of specified selector; only use this if response selector is HTTPp_MISC**
Selector Value	**Text**	**Text to set as the response selector value**
qi Append Value	**Longint**	**qi for whether new value should be appended to existing selector value**

The method *HTTPp_Set_Selector_x_s* will set the value of the specified HTTP response header selector. This can be useful if you want to build a custom HTTP response header.

HTTPp Selector Constant is the HTTPp Selector constant. This could also be an index used in a loop.

Selector Title is the title of the response selector to access. You should only use this value if the *HTTPp Selector Constant* is HTTPp_MISC.

Selector Value is the new text to set as the value of the selector. This text can either replace the old value, or be appended to it.

qi Append Value is a qi for whether the new value should replace the old value, or be appended to it. Set *qi Append Value* to zero (0) to replace the old value, or one (1) to append the new value to the old value.

_____

HTTPp_Set_Selector_z_s

HTTPp_Set_Selector_z_s (*HTTPp Selector Constant* ; *Selector Title* ; *Referenced BLOB* ; *qi Append Value*)

HTTPp_Set_Selector_z_s
    (
          -> *HTTPp Selector Constant* : **Longint**
          -> *Selector Title* : **Text**
          -> *Referenced BLOB* : **Pointer**
          -> *qi Append Value* : **Longint**
)

Parameter	Type	Description
HTTPp Selector Constant	**Longint**	**Index into HTTPp Selector Stack**
Selector Title	**Text**	**Title of specified selector; only use this if response selector is HTTPp_MISC**
Referenced BLOB	**Pointer**	**Reference to BLOB containing new value for selector**
qi Append Value	**Longint**	**qi for whether new value should be appended to existing selector value**

The method **HTTPp_Set_Selector_z_s** will set the value of the specified HTTP response header selector. This can be useful if you want to build a custom HTTP response header.

HTTPp Selector Constant is the HTTPp Selector constant. This could also be an index used in a loop.

Selector Title is the title of the response selector to access. You should only use this value if the *HTTPp Selector Constant* is HTTPp_MISC.

Referenced BLOB is a pointer to a BLOB to use as the new value of the selector. This BLOB can either replace the old value, or be appended to it.

qi Append Value is a qi for whether the new value should replace the old value, or be appended to it. Set *qi Append Value* to zero (0) to replace the old value, or one (1) to append the new value to the old value.

_____

HTTPp_Use_Default_Header

HTTPp_Use_Default_Header

HTTPp_Use_Default_Header

Parameter	Type	Description
None	None	None

The method *HTTPp_Use_Default_Header* will set the value of the response header selectors to their default values.

HTTPq Module

The HTTPq Module is for working with the HTTP request. The request is read and parsed for the developer by HTTP Server Deux. The HTTPq module contains accessor routines to access information from the HTTP request header, as well as any extra fields that were sent. Methods are also provided to conveniently handle uploaded files.

_____

HTTPq_ERROR

HTTPq_ERROR (*HTTPq Error Number* ; *Special Error Text* ; *Calling Method Name*)

HTTPq_ERROR
    (
            -> *HTTPq Error Number*: **Longint**
            -> *Special Error Text*: **Text**
            -> *Calling Method Name*: **Text**
)

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

The internal *HTTPq 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 HTTPq method which called the **HTTPq_ERROR** method.

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

_____

HTTPq_Extract_File_Header

HTTPq_Extract_File_Header (*Referenced Uploaded File BLOB* ; *Referenced Extracted Header BLOB*)

HTTPq_Extract_File_Header
    (
            -> *Referenced Uploaded File BLOB* : **Pointer**
            -> *Referenced Extracted Header BLOB* : **Pointer**
)

Parameter	Type	Description
Referenced Uploaded File BLOB	**Pointer**	**Reference to BLOB containing uploaded file with header**
Referenced Extracted Header BLOB	**Pointer**	**Reference to BLOB which will get the file header**

The method *HTTPp_Set_Selector_z_s* will set the value of the specified HTTP response header selector. This can be useful if you want to build a custom HTTP response header.

HTTPp Selector Constant is the HTTPp Selector constant. This could also be an index used in a loop.

Selector Title is the title of the response selector to access. You should only use this value if the *HTTPp Selector Constant* is HTTPp_MISC.

Referenced BLOB is a pointer to a BLOB to use as the new value of the selector. This BLOB can either replace the old value, or be appended to it.

qi Append Value is a qi for whether the new value should replace the old value, or be appended to it. Set *qi Append Value* to zero (0) to replace the old value, or one (1) to append the new value to the old value.

▨   HTTPq_Get_Field_Count_s

HTTPp_Get_Field_Count_s => *Number of Fields*

HTTPp_Get_Field_Count_s
        => *Number of Fields*

Parameter	Type	Description
Number of Fields	Longint	The number of elements in the HTTPq Request Fields Stack

The method *HTTPq_Get_Field_Count_s* returns the number of elements in the HTTPq Request Fields Stack. This would allow you to loop through the request fields stack to access fields by index.

Number of Fields is the number of elements in the HTTPq Request Fields Stack.

▨   HTTPq_Get_Field_Name_by_Index_s

HTTPq_Get_Field_Name_by_Index_s (*Request Fields Stack Index*) => *Request Field Name*

HTTPq_Get_Field_Name_by_Index_s
    (
        -> *Request Fields Stack Index* : **Longint**
)
        => *Request Field Name* : **Text**

Parameter	Type	Description
Request Fields Stack Index	Longint	Index into HTTPq Request Fields Stack
Request Field Name	Text	Field name from Request Fields Stack specified by the index

The method *HTTPq_Get_Field_Name_by_Index_s* will get the name of the specified field in the HTTPq Request Fields Stack. This is very useful if you are looping through the Request Fields Stack.

Request Fields Stack Index is the index to use to access the Request Field Stack.

Request Field Name is the field name from the Request Fields Stack.

_____

▓ HTTPq_Get_Field_Name_Count_s

HTTPq_Get_Field_Name_Count_s (*Request Field Name ; Request Type*)
=> *Request Field Name Count*

HTTPq_Get_Field_Name_Count_s
   (
        -> *Request Field Name* : **Text**
        -> *Request Type* : **Longint**
)
        => *Request Field Name Count* : **Longint**

Parameter	Type	Description
Request Field Name	Text	Name of request field to look up
Request Type	Longint	Type of request to count names for
Request Field Name Count	Longint	Number of field names matching the specified field name and request type

The method *HTTPq_Get_Field_Name_Count_s* will get the number of elements with the same field name that were in the specified request type. This is useful for scrolling lists with multiple selections, where each selection will be an element in the HTTPq Request Field Stack, each having the name of the scrolling list. Also,

in some cases, there may be multiple checkboxes with the same name but different values.

Request Field Name is the field name to look for in the Request Field Stack. This could also be a partial name, ending with the wildcard (@).

Request Type is the type of request (GET or POST) to count field names for. See the HTTPq Request Types constant group for constants to use for this parameter.

Request Field Name Count is the number of times the *Request Field Name* appears with the *Request Type* in the Request Fields Stack.

_____

📑 HTTPq_Get_Field_Type_s

HTTPq_Get_Field_Type_s (*Request Field Name ; Nth Occurrence*) => *Request Field Type*

HTTPq_Get_Field_Type_s
   (
         -> *Request Field Name* : **Text**
         -> *Nth Occurrence* : **Longint**
)
         => *Request Field Type* : **Longint**

Parameter	Type	Description
Request Field Name	**Text**	**Name of request field to look up**
Nth Occurrence	**Longint**	**Nth occurrence of the field name to look up**
Request Field Typet	**Longint**	**Request type of the specified request field**

The method **HTTPq_Get_Field_Type_s** will get the request type of the specified field in the HTTPq Request Field Stack. This allows the developer to determine if the specified field was a GET or POST field.

Request Field Name is the field name to look for in the Request Field Stack. This could also be a partial name, ending with the wildcard (@).

Nth Occurrence is the Nth occurrence of the field name in the HTTPq Request Fields Stack to look up.

Request Type is the type of request (GET or POST) the specified request field was in. See the HTTPq Request Types constant group for constants to use for this parameter.

_____

HTTPq_Get_Field_Value_x_s

HTTPq_Get_Field_Value_x_s (*Request Field Name ; Nth Occurrence ; Request Type ; Max Text Length*) => *Request Field Value*

HTTPq_Get_Field_Value_x_s
    (
        -> *Request Field Name* : **Text**
        -> *Nth Occurrence* : **Longint**
        -> *Request Type* : **Longint**
        -> *Max Text Length* : **Longint**
)
        => *Request Field Value* : **Text**

Parameter	Type	Description
Request Field Name	**Text**	**Name of request field to look up**
Nth Occurrence	**Longint**	**Nth occurrence of the field name to look up**
RequestField Typet	**Longint**	**Request type of the field to look up**
Max Text Length	**Longint**	**Max length of text to return**
Request Field Value	**Text**	**Value of named request field**

The method **HTTPq_Get_Field_Value_x_s** will get the value of the specified field from the HTTPq Request Field Stack.

Request Field Name is the field name to look for in the Request Field Stack.

Nth Occurrence is the Nth occurrence of the field name in the HTTPq Request Fields Stack to look up.

Request Field Type is the type of request (GET or POST) the specified request field in. See the HTTPq Request Types constant group for constants to use for this parameter.

Max Text Length is the maximum length of the text value to return. If the length of the request field value is less than the length specified, the entire value will be returned. If the length of the request field value is greater than *Max Text Length*, the value will be truncated.

Request Field Value is the specified request field value, returned as text.

_____

HTTPq_Get_Field_Value_z_s

HTTPq_Get_Field_Value_z_s (*Request Field Name ; Nth Occurrence ; Request Type ; Referenced BLOB*)

HTTPq_Get_Field_Value_z_s
    (
        -> *Request Field Name* : **Text**
        -> *Nth Occurrence* : **Longint**
        -> *Request Type* : **Longint**
        -> *Referenced BLOB* : **Pointer**
)

Parameter	Type	Description
Request Field Name	**Text**	**Name of request field to look up**
Nth Occurrence	**Longint**	**Nth occurrence of the field name to look up**
RequestField Typet	**Longint**	**Request type of the field to look up**

Referenced BLOB	**Pointer**	**Pointer to a BLOB which will receive the value of the specified request field**

The method *HTTPq_Get_Field_Value_z_s* will get the value of the specified field from the HTTPq Request Field Stack.

Request Field Name is the field name to look for in the Request Field Stack.

Nth Occurrence is the Nth occurrence of the field name in the HTTPq Request Fields Stack to look up.

Request Field Type is the type of request (GET or POST) the specified request field in. See the HTTPq Request Types constant group for constants to use for this parameter.

Referenced BLOB is a pointer to a BLOB which will receive the entire request field value.

HTTPq_Get_Files_Count_s

HTTPp_Get_Files_Count_s => *Number of Files*

HTTPp_Get_Files_Count_s
> => *Number of Files*

Parameter	Type	Description
Number of Files	Longint	The number of uploaded files in the HTTPq Request Fields Stack

The method *HTTPq_Get_Files_Count_s* returns the number of uploaded files in the HTTPq Request Fields Stack.

Number of Files is the number of uploaded files in the HTTPq Request Fields Stack.

HTTPq_Get_File_Properties

HTTPq_Get_File_Properties (*Referenced File Upload Header ; Referenced File Name ; Referenced Content-Type ; Referenced Transfer-Encoding*)

HTTPq_Get_File_Properties
    (
            -> *Referenced File Upload Header* : **Pointer**
            -> *Referenced File Name* : **Pointer**
            -> *Referenced Content-Type* : **Pointer**
            -> *Referenced Transfer-Encoding* : **Pointer**
)

Parameter	Type	Description
Referenced File Upload Header	**Pointer**	**Pointer to BLOB containing uploaded file header**
Referenced File Name	**Pointer**	**Pointer to text variable to contain uploaded file's name**
Referenced Content-Type	**Pointer**	**Pointer to text variable to contain uploaded file's content-type**
Referenced Transfer-Encoding	**Pointer**	**Pointer to text variable to contain uploaded file's transfer-encoding**

The method *HTTPq_Get_File_Properties* will extract file properties from the uploaded file's header. This convenience method makes it simple to get the uploaded file's name and other information about uploaded files.

Referenced File Upload Header is a referenced BLOB containing the uploaded file's header. The uploaded file's header can be extracted from the file with the method *HTTPq_Extract_File_Header*.

Referenced File Name is a reference to a text variable to receive the uploaded file's name.

Referenced Content-Type is a reference to a text variable to receive the uploaded file's content type. Pass a NULL pointer to skip this value.

Referenced Transfer-Encoding is a reference to a text variable to receive the uploaded file's transfer encoding. Pass a NULL pointer to skip this value.

▓ **HTTPq_Get_Request_Body_z**

HTTPq_Get_Request_Body_z (*Referenced BLOB*)

HTTPq_Get_Request_Body_z
   (
       -> *Referenced BLOB* : **Pointer**
)

Parameter	Type	Description
Referenced BLOB	Pointer	Pointer to BLOB to receive full request body

The method *HTTPq_Get_Request_Body_z* will return the entire HTTP request body into the referenced BLOB.

Referenced BLOB is a referenced BLOB to receive the full HTTP request body.

▓ **HTTPq_Get_Request_Header_z**

HTTPq_Get_Request_Header_z (*Referenced BLOB*)

HTTPq_Get_Request_Header_z
   (
       -> *Referenced BLOB* : **Pointer**
)

Parameter	Type	Description
Referenced BLOB	Pointer	Pointer to BLOB to receive full request header

The method *HTTPq_Get_Request_Body_z* will return the entire HTTP request header into the referenced BLOB.

Referenced BLOB is a referenced BLOB to receive the full HTTP request header.

▧ HTTPq_Get_Selector_Count_s

HTTPp_Get_Selector_Count_s => *Number of Selectors*

HTTPp_Get_Selectors_Count_s
   => *Number of Selectors*

Parameter	Type	Description
Number of Selectors	Longint	The number of elements in the HTTPq Request Selectors Stack

The method *HTTPq_Get_Selector_Count_s* returns the number of elements in the HTTPq Request Selectors Stack.

Number of Selectors is the number of elements in the HTTPq Request Selectors Stack.

▧ HTTPq_Get_Selector_Title_s

HTTPq_Get_Selector_Title_s (*Request Selector*) => *Request Selector Title*

HTTPq_Get_Selector_Title_s
   (
      -> *Request Selector* : Longint
)
      => *Request Selector Title* : Text

Parameter	Type	Description
Request Selector	Longint	Request selector
Request Selector Title	Text	Title of specified request selector

The method *HTTPq_Get_Selector_Title_s* will get the title of the specified request selector. This is very

useful if you are looping through the HTTPq Request Selectors Stack.

Request Selector is a code identifying the selector title to retrieve. See the HTTPq Selectors custom constant group for constants to use to specify selectors.

Request Selector Title is the specified selector title.

_____

▓ HTTPq_Get_Selector_x_s

HTTPq_Get_Selector_x_s (*Request Selector ; Max Text Length*) => *Request Selector Value*

HTTPq_Get_Selector_x_s
> (
>> -> *Request Selector* : **Text**
>> -> *Max Text Length* : **Longint**
>)
>> => *Request Selector Value* : **Text**

Parameter	Type	Description
Request Selector	**Longint**	**Request Selector**
Max Text Length	**Longint**	**Max length of text to return**
Request Selector Value	**Text**	**Value of specified request selector**

The method **HTTPq_Get_Selector_x_s** will get the value of the specified request selector from the HTTPq Request Selectors Stack.

Request Selector is a code identifying the selector title to retrieve. See the HTTPq Selectors custom constant group for constants to use to specify selectors.

Max Text Length is the maximum length of the text value to return. If the length of the request field value is less than the length specified, the entire value will be returned. If the length of the request field value is greater than *Max Text Length*, the value will be truncated.

Request Selector Value is the specified request selector value, returned as text.

_____

HTTPq_Get_Selector_z_s

HTTPq_Get_Selector_z_s (*Request Selector ; Referenced BLOB*)

HTTPq_Get_Selector_z_s
    (
            -> *Request Selector* : **Longint**
            -> *Referenced BLOB* : **Pointer**
)

Parameter	Type	Description
Request Selector	**Longint**	**Request selector**
Referenced BLOB	**Pointer**	**Pointer to a BLOB which will receive the value of the specified request selector**

The method *HTTPq_Get_Selector_z_s* will get the value of the specified request selector from the HTTPq Request Selectors Stack.

Request Selector is a code identifying the selector title to retrieve. See the HTTPq Selectors custom constant group for constants to use to specify selectors.

Referenced BLOB is a pointer to a BLOB which will receive the entire specified request selector value.

_____

HTTPq_Parse_File_Header

HTTPq_Parse_File_Header (*Referenced File Upload Header ; Referenced Names Array ; Referenced Values Array*) => *Parsed Stack Size*

HTTPq_Parse_File_Header
    (
            -> *Referenced File Upload Header* : **Pointer**

-> *Referenced Names Array* : **Pointer**
        -> *Referenced Values Array* : **Pointer**
)

        => *Parsed Stack Size* : **Longint**

Parameter	Type	Description
Referenced File Upload Header	**Pointer**	**Pointer to BLOB containing uploaded file header**
Referenced Names Array	**Pointer**	**Pointer to text array to receive names of file header attributes**
Referenced Values Array	**Pointer**	**Pointer to text array to receive values of file header attributes**
Parsed Stack Size	**Longint**	**Number of elements in NVP arrays**

The method ***HTTPq_Parse_File_Header*** will parse out the file attributes from the header from an uploaded file. The names and values of the file attributes will be put into the referenced arrays.

Referenced File Upload Header is a referenced BLOB containing the uploaded file's header. Use the method ***HTTPq_Extract_File_Header*** to extract the file header from the uploaded file.

Referenced Names Array is a referenced text array that will receive the names of the file attributes from the *Referenced File Upload Header*.

Referenced Values Array is a referenced text array that will receive the values of the file attributes from the *Referenced File Upload Header*.

Parsed Stack Size is the number of attributes parsed out of the *Referenced File Upload Header* into the referenced NVP (name-value pair) arrays. This will be equal to the size of the NVP arrays after running this method.

_____

HTTPq_qi_Field_is_File_s

HTTPq_qi_Field_is_File_s (*Request Field Name* ; *Nth Occurrence*) => *qi Field Is File*

HTTPq_qi_Field_is_File_s
    (
           -> *Request Field Name* : **Text**
           -> *Nth Occurrence* : **Longint**
)
           => *qi Field Is File* : **Text**

Parameter	Type	Description
Request Field Name	**Text**	**Name of request field to look up**
Nth Occurrence	**Longint**	**Nth occurrence of the field name to look up**
qi Field is File	**Longint**	**qi for whether specified request field is an uploaded file or not**

The method **HTTPq_qi_Field_is_File_s** will determine if the specified request field is an uploaded file.

Request Field Name is the field name to look for in the Request Field Stack.

Nth Occurrence is the Nth occurrence of the field name in the HTTPq Request Fields Stack to look up.

qi Field is File is a qi for whether the specified request field is an uploaded file. qi Field is File will be set to zero (0) if the specified request field is not an uploaded file, or will be set to one (1) if the specified field is an uploaded file.

HTTPsd Module

The HTTPsd Module handles the server aspects of HTTP Server Deux. Use the methods in the HTTPsd module to define or clear HTTP Server Deux Services, as well as pause and resume the HTTP Server Deux server.

_____

HTTPsd_Clear_Services_All_s

HTTPsd_Clear_Services_All_s

HTTPsd_Clear_Services_All_s

Parameter	Type	Description

The method *HTTPsd_Clear_Services_All_s* will clear all services from the HTTPsd 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 services.

_____

HTTPsd_Clear_Service_s

HTTPsd_Clear_Service_s (*Local Port*)

HTTPsd_Clear_Service_s
    (
        -> *Local Port* : **Longint**
)

Parameter	Type	Description
Local Port	Longint	Local Port of Service to be cleared

The method *HTTPsd_Clear_Service_s* will clear the specified service from the HTTPsd 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.

Local Port is the port number assigned when creating the service with *HTTPsd_Create_Service_s*.

_____

HTTPsd_Create_Service_s

HTTPsd_Create_Service_s (*IP Address* ; *Local Port* ; *Number of Listeners* ; *Handler Method Name* ; *Protocol* ; *SSL Certificate Full Path* ; *SSL Private Key Full Path* ; *SSL Private Key Password*) => *qi Service Created*

HTTPsd_Create_Service_s
    (
        -> *IP Address* : **Longint**
        -> *Local Port* : **Longint**
        -> *Number of Listeners* : **Longint**
        -> *Handler Method Name* : **String [32]**
        -> *Protocol* : **Longint**
        -> *SSL Private Key Full Path* : **Text**
        -> *SSL Certificate Full Path* : **Text**
        -> *SSL Private Key Password* : **Text**
)
        => *qi Service Created* : **Longint**

Parameter	Type	Description
IP Address	**Longint**	**IP Address to listen on**
Local Port	**Longint**	**Port of local host to accept connections on**
Number of Listeners	**Longint**	**Number of TCP listeners to open for this service**
Handler Method Name	**String [32]**	**Method to be executed when a connection is established**

Protocol	**Longint**	**TCPd Protocol to be used for this service's listeners**
SSL Certificate Full Path	**Text**	**Full path to SSL certificate file (ITK v2.5.x only)**
SSL Private Key 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 HTTPsd Services Stack**

The method **HTTPsd_Create_Service_s** creates a new service in the HTTPsd 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.

IP Address is the local IP address for this service to use. This is useful when the server is configured with multiple IP addresses. Setting this parameter to zero (0) will 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 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.

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.

Handler Method Name is the name of the method to be executed when a connection is established. This method should build the information to be sent back to the web browser.

Protocol is the protocol to be used with the new TCP communications stream. This value has no affect on the

new TCP communications stream but is stored in the
TCPd Streams Stack for later reference. Valid values for
the *Protocol* parameter include all of the protocol values
in the constants group *TCPd_Protocols* included with the
TCP Deux component (documented separately in this
manual).

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

_____

▦   HTTPsd_ERROR

HTTPsd_ERROR (*HTTPsd Error Number* ; *Special Error Text ; Calling
          Method Name*)

HTTPsd_ERROR
     (

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

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

The internal *HTTPsd 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 HTTPsd method which called the **HTTPsd_ERROR** method.

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

_____

HTTPsd_Pack_Transaction

HTTPsd_Pack_Transaction (*Referenced BLOB*)

HTTPsd_Pack_Transaction
(
    -> *Referenced BLOB* : **Pointer**
)

Parameter	Type	Description
Referenced BLOB	Pointer	Referenced BLOB to pack transaction into

The method **HTTPsd_Pack_Transaction** will pack the HTTP request and response info into a BLOB.

Note: this is available for compatibility with Adjunct Components and should not be called by web developers.

Referenced BLOB is a reference to a BLOB to receive information about the current HTTP transaction (request and response).

Note: this method was added within the component as of v1.0.0b02.

_____

HTTPsd_Pause_Server

HTTPsd_Pause_Server

HTTPsd_Pause_Server

Parameter	Type	Description

The method **HTTPsd_Pause_Server** pauses the HTTP server. All listeners are left open, and an HTTP 503 response is returned for any request. To resume the HTTP Server, call the method **HTTPsd_Resume_Server**.

_____

HTTPsd_qi_AdjC_Installed_s

HTTPsd_qi_AdjC_Installed_s (*HTTPsd Adjunct Component ID*) => *qi Installed*

HTTPsd_qi_AdjC_Installed_s
    (
        HTTPsd Adjunct Component ID : **Longint**

)

      => *qi Installed* : **Longint**

Parameter	Type	Description
HTTPsd Adjunct Component ID	**Longint**	**ID of adjunct component to check for**
qi Installed	**Longint**	**qi for whether** *HTTPsd Adjunct Component* **is installed**

The method ***HTTPsd_qi_AdjC_Installed*** returns an indicator for whether the HTTPsd Adjunct Component is installed. Check the constant group HTTPsd Adjunct Component IDs for valid adjunct components.

HTTPsd Adjunct Component ID is the ID of the adjunct component to check for. Valid adjunct component IDs can be found in the HTTPsd Adjunct Component IDs constant group.

qi Installed is the indicator for whether the HTTPsd Adjunct Component is currently installed. *qi Installed* will be set to zero (0) if the adjunct component is not installed and will be set to one (1) if the adjunct component is not installed.

_____

HTTPsd_qi_Server_Paused

HTTPsd_qi_Server_Paused => *qi Server Paused*

HTTPsd_qi_Server_Paused
      => *qi Server Paused*: **Longint**

Parameter	Type	Description
qi Server Paused	**Longint**	**qi for whether the HTTPsd Server is currently paused**

The method ***HTTPsd_qi_Server_Paused*** returns an indicator for whether the HTTPsd Server is currently paused. When the HTTPsd Server is paused, an HTTP 503 response is returned for any request.

qi Server Paused is the indicator for whether the HTTPsd Server is currently paused. *qi Server Paused* will be set to zero (O) if the server is not paused and will be set to one (1) if the server is paused.

_____

HTTPsd_Register_AdjC_s

HTTPsd_Register_AdjC_s (*HTTPsd Adjunct Component ID*)

HTTPsd_Register_AdjC _s
    (
        HTTPsd Adjunct Component ID : **Longint**
)

Parameter	Type	Description
HTTPsd Adjunct Component ID	**Longint**	**ID of adjunct component to register**

The method ***HTTPsd_Register_AdjC*** registers an HTTPsd Adjunct Component for use by HTTP Server Deux. Check the constant group HTTPsd Adjunct Component IDs for valid adjunct components.

Note: this is available for compatibility with Adjunct Components and should not be called by web developers.

HTTP Adjunct Component ID is the adjunct component ID.

Note: this method was added within the component as of v1.0.0b02.

_____

HTTPsd_Resume_Server

HTTPsd_Resume_Server

HTTPsd_Resume_Server

Parameter	Type	Description

The method *HTTPsd_Resume_Server* resumes the HTTPsd Server. The handler method specified for the services will be executed when a connection is made. To pause the HTTPsd Server, call the method *HTTPsd_Pause_Server*. To check to see if the server is paused, call the method *HTTPsd_qi_Server_Paused*.

_____

HTTPsd_Send_BouncePage

HTTPsd_Send_BouncePage

HTTPsd_Send_BouncePage

Parameter	Type	Description

The method *HTTPsd_Send_BouncePage* displays all selectors in the HTTP request as the response page. This is very useful to see exactly what is being sent to the server, and how it is being parsed.

_____

HTTPsd_Set_Server_Prefs

HTTPsd_Set_Server_Prefs (*Send Timeout*)

HTTPsd_Set_Server_Prefs
   (
       -> *Send Timeout* : **Longint**
)

Parameter	Type	Description
Send Timeout	Longint	Time in seconds to wait when sending data before connection is closed

The method *HTTPsd_Set_Server_Prefs* will set preferences used by the HTTPsd Server. These preferences are global, and affect the operation of the

HTTPsd Server. They are not specific to any individual service.

Note: These preferences must be set before creating HTTP services with *HTTPsd_Create_Service_s*.

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

Version History

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

HTTP Server Deux v1.0.0b02

released 20020115

Changes:

The Windows copies of the Affix documents for HTTPsd were missing all '4DK#' resources. This has been fixed.

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

Fixed a bug in the parsing and retrieval of the realm password from the HTTP request header.

Fixed a bug in the parsing and retrieval of cookies from the HTTP request header.

Fixed a bug in the method **HTTPq_Get_Field_Name_Count_s** in which the routine could get caught in an infinite loop.

Added support for the Post Process Handler for all HTTPsd services.

Added the new constants group <u>HTTPsd Adjunct Component IDs</u>.

Added *HTTPsd Adjunct Components Stack* and support throughout component.

Updated 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 functionality to save cookie values even when response code is not 200.

Fixed bug in building of response header where CRLF was blindly added to the end of every line; if a header value already ended with CRLF, this would make a double CRLF, thus ending the header even though other header lines may not be added yet.

Fixed the method **HTTPp_Set_Cookie_Selector_s** to not append a CRLF at the end of the cookie line. Instead, it checks if any cookies have been set yet, and if so, prefixes the new cookie lines with CRLF to end the previous cookie line.

Added request transaction variables for storing DTS and milliseconds of when transaction was received.

Added response transaction variables for storing DTS of when response sent completed and total bytes sent for complete response.

Renamed affix document on Mac from "Affix_HTTPsd" to "Affix_HTTPs_Deux_MacOS9.4DX".

Renamed affix document on Windows from "Afx_HTTPsd" to "Affix_HTTPs_Deux".

Added protected methods:
> **HTTPp_Get_ResponseCode**
> **HTTPsd_Pack_Transaction**
> **HTTPsd_qi_AdjC_Installed_s**
> **HTTPsd_Register_AdjC_s**

HTTP Server Deux v1.0.0b01
released 20011012

Changes:

First public beta release of the component.

HTTP Server Deux Error Codes

asd

Index

asd