4D Internet Commands

4D Internet Commands

Preface

The 4D Internet Commands empower users of 4D with a robust set of communication utilities capable of working in either a Local or Wide area network. The ultimate expression of this explosion of connectivity is known colloquially as "The Internet". The last few years has produced phenomenal growth in the number of people and companies gaining access to the Internet. As the volume of people with Internet access increases, the need to be "on the net" is felt more and more each day by those in the business community.

The suite of commands provided by 4D Internet Commands gives database developers access to many key elements of the Internet. The SMTP commands contain tools to automate e-mail delivery from a database to any list of people. Similarly, the POP3 and IMAP commands can retrieve mail from any number of mailboxes for storage within the database, re-routing, auto-reply or remote-search execution. The FTP commands enable the user to transfer files to/from remote systems or to obtain directory listings of files on FTP volumes. And both the TCP and UDP commands provide developers with the low-level tools enabling them to accomplish any internet-related task.

SMTP (Simple Mail Transfer Protocol) is the primary mail transfer protocol used over the Internet. 4D Internet Commands allow users to quickly build and send mail via a SMTP server. Mail creation and delivery can be as simple as a single command. If your mail delivery needs are more complex, every aspect of the message header, body and attachments can be controlled to affect its delivery. Since Internet mail addressing provides for delivery to such services as CompuServe, America Online, eWorld, etc. you are able to reach virtually anyone with an e-mail account. Other examples of how the suite of SMTP commands could be used are:

- Automation of database report delivery
- Creation of an automatic mail forwarding database
- Group mail-list management
- Store-and-forward remote database updates & synchronizations

Along with its SMTP commands, 4D Internet Commands also contains commands that will connect either to POP3 (Post Office Protocol, Version 3) or IMAP (Internet Message Access Protocol) mail servers for retrieval of mail messages and encoded attachments. Since the suite of SMTP, POP3 and IMAP commands conforms to the MIME standard for multiple message enclosures, binary attachments can easily be downloaded and saved.

The commands also provide users with the ability to encode attachments in several different ways such as: Binhex, Base64, AppleSingle, AppleDouble...

The FTP (File Transfer Protocol) commands provide a very easy-to-use mechanism for communicating with an FTP server to send/receive text or binary files. Commands within the FTP suite can obtain directory listings of files, enabling the database developer to create navigable interfaces to remote volumes of files. The FTP commands can easily be used in document-tracking applications without requiring client applications to mount remote volumes directly.

Transmission Control Protocol/Internet Protocol, (TCP/IP) is the primary protocol used for sending and receiving data over the Internet. 4D Internet Commands contains several commands for sending and receiving raw TCP packets. The TCP set of commands provides developers with the essential tools to build and control their own Internet communications. In addition, the *TCP_Open* command allows connection using a SSL (Secured Socket Layer) protocol.

Some examples are:

- Build your own telnet interface
- Execute shell commands on remote machines
- Retrieve documents from the World Wide Web
- Search through numerous on-line databases
- Handle database synchronizations with remote servers
- Federal Express and UPS package tracking
- Connect to a Web server in Https.

Note: For greater flexibility, 4D Internet commands let you pass a POP3, IMAP or FTP connection reference directly to low-level TCP commands and vice versa. For more information, refer to the **Low Level Routines, Overview** section.

The UDP protocol is an easy-to-implement protocol for sending data that is faster and simpler than TCP, but with lower reliability since it does not allow delivery verification, error-checking or recovery of data that was improperly delivered.

Installation and Software Requirements

Installation

The “4D Internet Commands” plug-in is integrated in 4D in the same way as other plug-ins.

4D Internet Commands becomes automatically available when you install a 4D product since the plug-in is automatically installed in the **PlugIns** folder of your application.

For more information on plug-in installation and configuration, please refer to the 4D Product Line Installation Guide.

Software Requirements

4D Internet Commands require the same system configuration as required for 4D. For more information, refer to the 4D's *Installation Guide*.

4D (Mac and Windows)

The 4D Internet Commands 64-bit version must be used with the 64-bit version of 4D Server:

- Windows: 4D Server 64-bit version is available starting with version 12.1
- OS X: 4D Server 64-bit version for OS X is available starting with version 14 R3

Network access

In order to use the suite of commands in 4D Internet Commands, you must have access to a network that supports TCP/IP.

SSL

4D Internet Commands allows you to use the secure protocol with commands for sending messages and connecting with the e-mail servers. No special configuration is needed to use this protocol in 4DIC.

Note: The implementation of SSL/TLS in 4DIC uses the "implicit method".

Domain Name Server

For many of the 4D Internet Commands, it is necessary to have access to a domain name server. For more information, please refer to your network administrator.

SMTP Mail Server

In order to send mail using the set of SMTP commands, it is necessary for the sender to have access to a SMTP mail server, which will forward the message to a POP3 mail server.

POP3 Mail Server

In order to use the POP3 commands, you must have an account on a POP3 mail server.

IMAP Mail Server

In order to use the IMAP commands, you must have an account on an IMAP mail server.

Glossary and Terminology

This section defines many of the references made throughout the manual. The definitions are simplistic and are meant mainly for those unfamiliar with the references. The Terminology section pertaining to "Parameter Formats" provides details on the formatting expectations of 4D Internet Commands common parameters.

NIC: "Network Information Center". For the most part, the Internet is an unregulated entity. There is no centralized authority or control over its use or growth. However, there are some basic administrative needs such as domain name and IP address assignments that could only be effectively carried out if controlled by a single agency. The NIC is the group responsible for such administrative tasks.

RFC: "Request for Comments." Most of the 4D Internet Commands are based upon standards defined to handle Internet communication. The standard methodologies, descriptions and protocols used throughout the Internet are defined within documents known as RFCs. **Appendix D, Additional Information...** contains references to some WWW sites with pointers to many of the RFC documents. The 4D Internet Commands package does its best to protect you from a need to reference these documents, though anyone programming their own communications via the low-level TCP routines should become familiar with them.

TCP/IP Addresses, Host Names and Domain Names: An IP address is a reference to a **specific** machine somewhere out in the world. The IP address is formatted as a string containing four numeric values separated by periods (i.e. "207.94.15.3"). Each numeric part of the address can contain a value between zero and 255. By applying some mathematical functions to an IP address, its value can be squeezed down into an equivalent Long Integer number, which this document will refer to as the **ip_LongInt**.

In order for a site (i.e. a Company, College, etc.) to put their computers on the Internet, some assurances must be taken that their IP addresses won't conflict with other machines on the network. Institutions (and often individuals) will register their site with the **NIC** in order to obtain a **Domain Name**. **Domain Names** provide a system of easy-to-remember Internet addresses, which can be translated by the Domain Name System (DNS) into the numeric addresses (Internet Protocol [IP] numbers) used by the network. This system allows a more readable format such as "www.4D.com" or "ftp.4D.com".

Domain Name = "4D.com"

Host Name (Name of a computer)	=	IP address	=	ip_LongInt
"www.4D.com"	=	"207.94.15.3"	=	-815919357

The relationship between a **Host name** and its corresponding IP address is stored in a database known as a DNS (Domain Name System). These servers communicate with one another to exchange any new or changed data in the domain name lists throughout the world. The TCP/IP control panel provides a means to 'point' your computer to a DNS, which will then resolve all domain name references you use.

It is important to understand that all domain name servers have a corresponding IP address. However, not all IP addresses have a corresponding domain name server. Also, a "Mail Address" such as "jsmith@4D.com" does not reference that person's specific computer or IP address. The mail address would direct its delivery to the machine with the IP address represented by resolving the domain "4D.com". The mail would be delivered to the POP3 server running on that machine, which would then hold the mail for its user named "jsmith".

Domain Name: The Domain Name is an addressing construct used for identifying and locating computers on the Internet. Domain names provide a system of easy-to-remember Internet addresses, which can be translated by the Domain Name System (DNS) into the numeric addresses (Internet Protocol [IP] numbers) used by the network. A domain name is hierarchical and often conveys information about the type of entity using the domain name. A domain name is simply a label that represents a domain, which is a subset of the total domain name space. Domain names at the same level of the hierarchy must be unique, for example there can be only one *com* at the top level of the hierarchy, and only one *4D.com* at the next level of the hierarchy. If your organization's name is "CompanyName" you could register the domain name "CompanyName.com" and your e-mail address could be "UserName@CompanyName.com". Your customers would also be able to access your organization's web site by visiting "www.companyName.com" with their Web browser.

Domain Name System (DNS): A distributed database of information that is used to translate domain names, which are easy for humans to remember and use, into Internet Protocol (IP) numbers, which are what computers need to find each other on the Internet. People working on computers around the globe maintain their specific portion of this database, and the data held in each portion of the database is made available to all computers and users on the Internet. The DNS comprises computers, data files, software, and people working together.

Encoding: Encoding converts a file from one format to another so that a file can be moved across different computer systems which may not all support the same character sets. The most common form of encoding is binary-hexadecimal (Binhex) encoding. Binhex encoding is the default encoding option for any attachments that you add to messages. While encoding creates a new file that is larger than the original, it converts the data fork, resource fork, and Finder information into a character file which can easily be sent as an attachment. 4D Internet Commands support the most common encoding methods, including Binhex, Base64, AppleSingle, AppleDouble, UUEncode and MacBinary.

Encryption: Encryption is used to intentionally scramble the contents of messages. Messages are encrypted using an external encryption program such as PGP, for the sole purpose of increasing the privacy of messages. The encrypted text must then be decrypted before it can be read. 4D Internet Commands do **NOT** provide any means for encrypting text.

Compression: Is used as a means of reducing the space taken up by a file. In order to compress a file, the file must be run through an application such as Stuffit Deluxe™ Compact Pro™ or WinZip™. These files must then be decompressed using the application in order to return the file to its original format. When files are compressed using compression applications, it is common for those applications to append a suffix to the original name of the file. Below are some common suffixes and their respective applications.

Filename.SIT - Stuffit application
Filename.CPT - Compact Pro application
Filename.DD - Disk Doubler application
Filename.ZIP - Winzip application
Filename.SEA - Self Extracting Archive. These files are Macintosh stand-alone applications and will decompress themselves when the user double-clicks on them because application code for decompression is included. Due to the addition of this code, self-extracting archives are generally larger than if the file was created as Filename.SIT or Filename.CPT. However since the user doesn't need to have the compression application, this option may be advantageous to the end user.

It is important to remember that once compressed, a file still needs to be encoded prior to transmission to ensure that the file is properly transferred from machine to machine on its way to its ultimate destination.

Parameter Formats

The descriptions that follow provide details on the meaning and formatting of many key parameters used throughout this manual.

Parameter	Type		Description
hostName	String	→	Host name (Ex: "www.companyname.com")
			IP address (Ex: "204.118.90.2")
ip_LongInt	LongInt	→	Long Integer reference to a IP address
mailAddress	Text	→	Ex: "jsmith@4d.com"
addressList	Text	→	Ex: "jsmith@4d.com, jdupont@4d.fr" or
			"jsmith@4d.com"+Char(13)+"jdupont@4d.fr"
localPath	Text	→	- Document
			Mac: "My Hard Drive:4DDB:SalesDB:Report"
			Win: "C:¥MyDrive¥4DDB¥SalesDB¥Report.txt"
			- Directory
			Mac: "My Hard Drive:CoolStuff:" (Note trailing ":")
			Win: "C:¥MyDrive¥CoolStuff¥"
hostPath	Text	→	- Document
			"/usr/jsmith/reports/salesreport.txt"
			- Directory
			"/usr/jsmith/reports/"(Note trailing "/")
tcp_ID	LongInt	→	Reference to an open TCP session
smtp_ID	LongInt	→	Reference to a new mail message
pop3_ID	LongInt	→	Reference to an open POP3 session
imap_ID	Longint	→	Reference to an open IMAP connection
ftp_ID	LongInt	→	Reference to an open FTP session
udp_ID	Longint	→	Reference to a UDP socket
Function result	Integer	←	Error Code

hostName

The *hostName* is the host name or IP address, such as "dns.4d.com" or "204.118.90.2". Host names are resolved through a domain name system. The default and secondary domain name systems are typically set within the Control Panel of the installed TCP/IP driver. Any 4D Internet command requiring a *hostName* as a parameter will accept its value in either the name ("www.4d.com") or IP address ("204.118.90.2") format. The "name" format is always preferred since it buffers your application from ill effects due to hardware changes at remote sites.

ip_LongInt

All host names can be resolved via the methods described above to an IP address. Mathematical formulas can then be applied to the IP address to convert the value to a unique long integer number. Commands within the 'Special Functions' section such as **NET_NameToAddr** and **NET_AddrToName** automate this conversion process. This LongInt value is referred to as the *ip_LongInt* throughout this documentation. The LongInt value will only be of use in special circumstances by developers doing direct TCP communication. Some developers may also prefer to store the LongInt value of a domain name in order to conserve disk space compared to its string equivalent. However, for compatibility reasons with IPV6, 4D advises developers against using this feature.

mailAddress

The MailAddress is a fully qualified mail specification in the format "user_name@domain_name". Within this document, *mailAddress* refers to a single e-mail address. Any 4D Internet Commands parameter which can take more than one address will specifically state *addressList*. If a parameter has *mailAddress* as its only type, it can take one and only one mail address. The format of the *mailAddress* should be a full reference containing both the user name and domain name:

- "Felix Unger" <felix@pristine.com>
- oscar@slobs.com (Oscar Madison)

addressList

An *addressList* contains one or more e-mail addresses in the format of *mailAddress*, each delimited by a comma or carriage return. Carriage return delimiting is useful when providing users with a text field to enter or paste a number of addresses. The following three examples would each generate an acceptable *$AddressList* value:

```
$AddressList:="jsmith@4d.com"
$AddressList:="jsmith@4d.com,scott@4d.com,marcel@4d.fr"
For($i;1;Size of array(aAddresses))
   $AddressList:=$AddressList+aAddresses{$i}+Char(13)
End for
```

localPath

The *localPath* is the location of a file or directory on the users machine (Mac or Windows).

On a Macintosh, colons delimit items within folders. For example, the file "My Report" in the "Reports" folder on the hard drive titled "My Hard Drive" would have a pathname of "My Hard Drive:Reports:My Report". A directory specification on a Macintosh must end with a colon character. For example, if you wanted to place a new report in the same folder as the above example, you would refer to the directory as "My Hard Drive:Reports:". The decision to reference a File or Directory name is based on the context called for by the command.

A similar format is used under the Windows environment, except a backward slash "¥" is used instead of the colon.

Note: With the FTP protocol, 4D internet commands work with files whose names have a limited size. For more information, refer to the **File Transfer, Overview** section).

hostPath

The *hostPath* is the location of a file or directory on a computer running under the Unix operating system. In the Unix environment, directories are separated with slashes ("/"). For example, the file "report.txt" in the "reports" directory in the "4D" directory would be specified as "/4D/reports/report.txt". A directory pathname must end with a "/" character. Note that a full pathname begins with a "/" which represents the root of the volume.

Note: With the FTP protocol, 4D internet commands work with files whose names have a limited size. For more information, refer to the **File Transfer, Overview** section).

smtp_ID, pop3_ID, imap_ID, ftp_ID, tcp_ID

Throughout each section of 4D Internet Commands, references are made to an "ID" number in most of the commands. Each set of communication functions will establish their own "session" represented by a Long Integer "ID" number. Subsequent commands related to the open session will use this value to direct their effects down the proper channel.

The "ID" numbers obtained in each section (SMTP, POP3, IMAP, FTP, TCP, UDP) may not be passed as values to different sections. However, for greater flexibility, 4D Internet commands let you pass a POP3, IMAP or FTP connection reference directly to low-level TCP commands and vice versa. For more information, please refer to the **Low Level Routines, Overview** section.

Session Reference	Opened by	Closed by
tcp_ID	TCP_Open or TCP_Listen	TCP_Close
smtp_ID	SMTP_New	SMTP_Clear
pop3_ID	POP3_Login	POP3_Logout or POP3_VerifyID
imap_ID	IMAP_Login	IMAP_Logout or IMAP_VerifyID
ftp_ID	FTP_Login	FTP_Logout or FTP_VerifyID
udp_ID	UDP_New	UDP_Delete

Function result

All 4D Internet Commands (with the exception of **IT_ErrorText** and **IT_Version**) return an integer value as the result of the function. This integer contains any error number that the command needs to convey back to the 4D database. If a command is successful, a zero will be returned. Otherwise, an error code is returned. For more information about 4D Internet Commands error codes, please refer to **Appendix C, 4D Internet Commands Error Codes**.

IC Downloaded Mail

Downloading Mail, Overview

The set of commands prefixed by "MSG_" allows the user to manipulate mail messages which have been saved as local files using the *POP3_Download* or *IMAP_Download* command described in the previous section. Because the set of commands are fully MIME compliant, 4D Internet Commands provide the means for attachments to be extracted and/or decoded. For more information on the MIME standards, refer to RFC#1521, RFC#1522 and RFC#2045.

Once messages have been downloaded to local files, the commands in this section provide a variety of functions to manipulate the documents. These commands can obtain information about the message parts, separate the header detail from the message body, detect and extract attachments within the message as well as delete existing documents.

MSG_Charset

MSG_Charset (decodeHeaders ; bodyCharset) -> Function result

Parameter	Type		Description
decodeHeaders	Integer	⇒	-1 = Use current settings, 0 = Do not manage, 1 = Convert using the Mac OS char set if ISO-8859-1 or ISO-2022-JP, decode extended characters
bodyCharset	Integer	⇒	-1 = Use current settings, 0 = Do not manage, 1 = Convert using the Mac OS char set if ISO-8859-1 or ISO-2022-JP
Function result	Integer	⮐	Error code

Description

The *MSG_Charset* command allows automatic support of messages containing extended characters during processing with the MSG commands. If this command is not called or has parameters set to 0, 4D Internet Commands version 6.8.1 or higher will work the same way as version 6.5.x.

MSG_Charset allows, first of all, the setting of whether the extended characters header decoding must be managed and, second, the determining of whether the message body and header character set conversion must be managed.

This command is particularly useful for supporting extended characters included in message headers such as the "Subject" or electronic mail addresses (for instance, to decode an address such as *"=?ISO-8859-1?Q?Test=E9?= <test@n.net >"*).

The *decodeHeaders* parameter specifies how to manage header decoding and conversion while executing *MSG_FindHeader*. Default value is set to 0.

- -1: Use current settings;
- 0: Do not manage;
- 1: Headers are decoded if necessary. If decoded, and if the specified character set is ISO-8859-1 or ISO-2022-JP, headers are converted using Mac OS ASCII code or Shift-JIS respectively.

The *bodyCharset* parameter specifies how to manage message body character set conversion while executing the *MSG_GetBody* command. Default value is set to 0.

- -1: Use current settings;
- 0: Do not manage;
- 1: If the "Body-Content-Type" character set is set to ISO-8859-1 or ISO-2022-JP, the message body is converted using Mac OS ASCII code or Shift-JIS respectively.

Compatibility note (version 6.8.1): If the *MSG_Charset* command is not used and the *POP3_Charset* command has been used, the *MSG_FindHeader* and *MSG_GetBody* commands will take the *POP3_Charset* parameters into account. If *MSG_Charset* is used, the *POP3_Charset* parameters are ignored.

Example

Using version 6.8.1 or higher of 4D Internet Commands:

```
$Err:=MSG_Charset(1;1)
$Err:=MSG_FindHeader($msgfile;"From";$from)
$Err:=MSG_FindHeader($msgfile;"To";$to)
$Err:=MSG_FindHeader($msgfile;"Cc";$cc)
$Err:=MSG_FindHeader($msgfile;"Subject";$subject)
$Err:=MSG_MessageSize($msgfile;$HdrSize;$BdySize;$msgSize)
$Err:=MSG_GetBody($msgfile;0;$BdySize;$BodyContent).
```

MSG_Delete

MSG_Delete (fileName ; folder) -> Function result

Parameter	Type		Description
fileName	Text	→	Filename (path defaults to message folder)
folder	Integer	→	0 = Message Folder, 1 = Attachment Folder
Function result	Integer	↩	Error Code

Description

The *MSG_Delete* command deletes a local file.

fileName is the name of the file or the full path of the file to delete. If only a filename is given, the *folder* parameter is taken into account based on the following:

- *folder* = 0: the file resides in the message folder specified by *POP3_SetPrefs* or *MSG_SetPrefs*.
- *folder* = 1: the file resides in the attachment folder specified by *POP3_SetPrefs* or *MSG_SetPrefs*
  In both cases, if no *folder* is defined by *POP3_SetPrefs* or *MSG_SetPrefs*, the path will default to the folder containing the structure of the database (with 4D single-user) or in the 4D Client folder (with 4D Server).

Compatibility note (version 6.8.1): If the *MSG_SetPrefs* command is not used, the *POP3_SetPrefs msgFolder* and *attachFolder* parameters are used; if *MSG_SetPrefs* is used, the *POP3_SetPrefs* parameters are ignored.

Warning: This command will delete **ANY** file passed to it. Be very careful when using this command.

MSG_Extract

MSG_Extract (fileName ; decode ; attachmentPath ; enclosureList) -> Function result

Parameter	Type		Description
fileName	Text	→	Filename (path defaults to message folder)
decode	Integer	→	0 = No decoding, 1 = Decode if possible
attachmentPath	Text	→	FolderPath (path defaults to attachment folder)
enclosureList	String array	←	Enclosure filenames w/o FolderPath
Function result	Integer	↩	Error Code

Description

The *MSG_Extract* command extracts all attachments and puts them into the attachments folder.

fileName is the name of the file or the full path of the file of which to extract the attachments. If only a filename is given, the path will default to the folder set by *POP3_SetPrefs* or *MSG_SetPrefs* (see Compatibility note). If no folder has been specified, the path will default to the folder containing the structure of the database (with 4D single-user) or to the 4D Client folder (with 4D Server).

decode is an integer specifying whether to attempt to decode the attachment. A value of zero indicates no attempt should be made to decode the attachment(s). A value of 1 will attempt to decode the file if it has been encoded in one of the following ways: Binhex, AppleSingle, AppleDouble, or Base64.

attachmentPath is the FolderPath of where to save the attachment. If no FolderPath is specified, the file will be saved in the attachments folder as specified in *POP3_SetPrefs* or *MSG_SetPrefs* (see Compatibility note). If no FolderPath has been specified, the attachment will be saved in the same folder as the database structure.

Compatibility note (version 6.8.1): If the *MSG_SetPrefs* command is not used, the *POP3_SetPrefs msgFolder* and *attachmentPath* parameters are used; if *MSG_SetPrefs* is used, the *POP3_SetPrefs msgFolder* and *attachmentPath* parameters are ignored.

enclosureList is a text/string array which is returned containing the file names of each attachment. Only the document name will be returned in each array element, not the full pathname.

MSG_FindHeader

MSG_FindHeader (fileName ; headerLabel ; headerValue) -> Function result

Parameter	Type		Description
fileName	Text	→	Filename (path defaults to message folder)
headerLabel	String	→	Header label ("From:", "To:", "Subject:", etc.)
headerValue	Text	←	Value
Function result	Integer	↩	Error Code

Description

Given the *fileName* of a message document retrieved to disk by the *POP3_Download* or *IMAP_Download* command, the *MSG_FindHeader* command will search the header section for *headerLabel* and return the value assigned to the field into *headerValue*.

fileName is the name of the file or the full path of the file of which to extract the header information. If only a filename is given, the path will default to the folder set by *POP3_SetPrefs* or *MSG_SetPrefs* (see Compatibility note). If no folder has been specified by *POP3_SetPrefs*, the path will default to the folder containing the structure of the database (with 4D single-user) or to the 4D Client folder (with 4D Server).

Compatibility note (version 6.8.1): If the *MSG_SetPrefs* command is not used, the *POP3_SetPrefs msgFolder* parameter will be used; if *MSG_SetPrefs* is used, the *POP3_SetPrefs msgFolder* parameter is ignored.

headerLabel is a string containing the name of any header label. The *headerLabel* can reference any defined, user-defined or extended header such as "From:", "To:", "X-MyHeader", etc.

headerValue is a text variable where the command will return the value assigned to the specified header field. Since the *headerValue* parameter can include extended characters, you can automate their management using the *POP3_Charset* or *MSG_Charset* command.

Compatibility note (version 6.8.1): If the *MSG_Charset* command is not used, the *POP3_Charset bodyCharset* parameter will be used; if *MSG_Charset* is used, the *POP3_Charset bodyCharset* parameter is ignored.

MSG_GetBody

MSG_GetBody (fileName ; offset ; length ; bodyText) -> Function result

Parameter	Type		Description
fileName	Text	→	Filename (path defaults to message folder)
offset	Longint	→	Starting offset into message body (0 = start of body)
length	Longint	→	Number of characters
bodyText	Text	←	Body text (removes linefeeds if Prefs ON)
Function result	Longint	↩	Error Code

Description

The *MSG_GetBody* command returns just the text of the message. It will not include enclosure text and will strip all MIME headers.

fileName is the name of the file or the full path of the file of which to extract the body of the message. If only a filename is given, the path will default to the folder set by *POP3_SetPrefs* or *MSG_SetPrefs* (see Compatibility notes). If no folder has been specified by *POP3_SetPrefs*, the path will default to the folder containing the structure of the database (with 4D single-user) or to the 4D Client folder (with 4D Server).

offset is the character position within the source body information at which to begin the retrieval.

length is the number of characters to return.

bodyText receives the text of the message body. Since the *bodyText* parameter can include extended characters, you can automate their management using the *POP3_Charset* or *MSG_Charset* command (see Compatibility notes). This parameter takes the *stripLineFeed* parameter set by *POP3_SetPrefs* or *MSG_SetPrefs* into account (see Compatibility notes).

Compatibility notes (version 6.8.1):

- If the *MSG_SetPrefs* command is not used, the *POP3_SetPrefs msgFolder* and *stripLineFeed* parameters will be taken into account. If *MSG_SetPrefs* is used, the *POP3_SetPrefs msgFolder* and *stripLineFeed* parameters are ignored.
- If the *MSG_Charset* command is not used, the *POP3_Charset bodyCharset* parameter will be used; if *MSG_Charset* is used, the *POP3_Charset bodyCharset* parameter is ignored.

MSG_GetHeaders

MSG_GetHeaders (fileName ; offset ; length ; headerText) -> Function result

Parameter	Type		Description
fileName	Text	→	Filename (path defaults to message folder)
offset	Longint	→	Starting offset into headers (0 = start of header)
length	Longint	→	Number of characters
headerText	Text	←	Header text (removes linefeeds if Prefs ON)
Function result	Integer	↩	Error Code

Description

The *MSG_GetHeaders* command returns the raw text of the entire header section of the message. The header portion of a POP3 message is defined as the text from the beginning of the message to the first occurrence of two consecutive carriage return/line feed sequences.

fileName is the name of the file or the full path of the file of which to extract the header. If only a *fileName* is given, the path will default to the folder set by *POP3_SetPrefs* or *MSG_SetPrefs* (see Compatibility note). If no folder has been specified, the path will default to the folder containing the structure of the database (with 4D single-user) or to the 4D Client folder (with 4D Server).

offset is the character position within the source header information at which to begin the retrieval.

length is the number of characters to return. The length of the header section can be determined with *MSG_MessageSize*.

headerText receives the text of the header. This parameter takes the *stripLineFeed* parameter set by *POP3_SetPrefs* or *MSG_SetPrefs* into account.

Compatibility note (version 6.8.1): If the *MSG_SetPrefs* command is not used, the *POP3_SetPrefs msgFolder* parameter will be used; if *MSG_SetPrefs* is used, the *POP3_SetPrefs msgFolder* parameter is ignored.

MSG_GetMessage

MSG_GetMessage (fileName ; offset ; length ; rawText) -> Function result

Parameter	Type		Description
fileName	Text	⇒	Filename (path defaults to message folder)
offset	Longint	⇒	Starting offset into message file (0 = start of file)
length	Longint	⇒	Number of characters
rawText	Text	⇐	Raw text (ignores Prefs)
Function result	Integer	↩	Error Code

Description

The *MSG_GetMessage* command returns the raw text of the message regardless of enclosures. It does not strip MIME headers.

fileName is the name of the file or the full path of the file of which to extract the body of the message. If only a filename is given, the path will default to the folder set by *POP3_SetPrefs* or *MSG_SetPrefs* (see Compatibility note). If no folder has been specified, the path will default to the folder containing the structure of the database.

Compatibility note (version 6.8.1): If the *MSG_SetPrefs* command is not used, the *POP3_SetPrefs msgFolder* parameter will be used; if *MSG_SetPrefs* is used, the *POP3_SetPrefs msgFolder* parameter is ignored.

offset is the character position within the source message information at which to begin the retrieval.

length is the number of characters to return.

rawText receives the text of the entire message. The preference settings for linefeed stripping are ignored and no effort is taken to strip any attachment that may be embedded within the message body.

MSG_GetPrefs

MSG_GetPrefs (stripLineFeed ; msgFolder ; attachFolder) -> Function result

Parameter	Type		Description
stripLineFeed	Integer	←	0 = Do not strip CR/LF, 1 = Strip CR/LF
msgFolder	Text	←	Messages folder path ("" = no change)
attachFolder	Text	←	Attachments folder path ("" = no change)
Function result	Integer	↩	Error code

Description

The *MSG_GetPrefs* command returns the current preferences for the MSG commands.

The preferences are returned into the variables listed in the parameters.

stripLineFeed returns the current setting of the user’s preference for linefeed stripping.

msgFolder is a text variable which returns the local pathname to the default folder in which retrieved messages are stored.

attachFolder is a text variable which returns the local pathname to the default folder in which extracted attachments are stored

MSG_HasAttach

MSG_HasAttach (fileName ; attachCount) -> Function result

Parameter	Type		Description
fileName	Text	→	Filename (path defaults to message folder)
attachCount	Integer	←	Count of Attachments
Function result	Integer	↩	Error Code

Description

If the file has attachments, the *MSG_HasAttach* command returns in the integer *attachCount* the number of attachments. An attachment is any non-text MIME enclosure. If the message has no attachments, 0 is returned.

fileName is the name of the file or the full path of the file of which to check for attachments. If only a filename is given, the path will default to the folder set by *POP3_SetPrefs* or *MSG_SetPrefs* (see Compatibility note). If no folder has been specified, the path will default to the folder containing the structure of the database (with 4D single-user) or to the 4D Client folder (with 4D Server).

Compatibility note (version 6.8.1): If the *MSG_SetPrefs* command is not used, the *POP3_SetPrefs msgFolder* parameter is used; if *MSG_SetPrefs* is used, the *POP3_SetPrefs msgFolder* parameter is ignored.

attachCount is an integer value returned which specifies the number of attachments for *fileName*.

MSG_MessageSize

MSG_MessageSize (fileName ; headerSize ; bodySize ; msgSize) -> Function result

Parameter	Type		Description
fileName	Text	→	Filename (path defaults to message folder)
headerSize	Longint	←	Header size (subtracts linefeeds if Prefs ON)
bodySize	Longint	←	Body size (subtracts linefeeds if Prefs ON)
msgSize	Longint	←	Entire message or file size (ignores Prefs)
Function result	Integer	↩	Error Code

Description

Given the *fileName* of a message document retrieved to disk by the *POP3_Download* command, the *MSG_MessageSize* command returns information about the sizes of the various portions of the message.

fileName is the name of the file or the full path of the file of which to return message information. If only a filename is given, the path will default to the folder set by *POP3_SetPrefs* or *MSG_SetPrefs* (see Compatibility note). If no folder has been specified by *POP3_SetPrefs*, the path will default to the folder containing the structure of the database (with 4D single-user) or to the 4D Client folder (with 4D Server).

headerSize is the long integer variable returned containing the size of the header.
bodySize is the long integer variable returned containing the size of the body.
These two parameters take the *stripLineFeed* parameter set by *POP3_SetPrefs* or *MSG_SetPrefs* into account.

Compatibility note (version 6.8.1): If the *MSG_SetPrefs* command is not used, the *POP3_SetPrefs msgFolder* and *stripLineFeed* parameters will be taken into account if the *POP3_SetPrefs* command has been used previously; if *MSG_SetPrefs* is used, the *POP3_SetPrefs msgFolder* and *stripLineFeed* parameters are ignored.

msgSize is the long integer variable returned containing the size of the message.

MSG_SetPrefs

MSG_SetPrefs (stripLineFeed ; msgFolder ; attachFolder) -> Function result

Parameter	Type		Description
stripLineFeed	Integer	⇒	0 = Do not strip LineFeeds, 1 = Strip LineFeeds, -1 = No Change
msgFolder	Text	⇒	Messages folder path ("" = no change)
attachFolder	Text	⇒	Attachments folder path ("" = no change)
Function result	Integer	↩	Error code

Description

The *MSG_SetPrefs* command sets the preferences for all MSG commands.

stripLineFeed is an integer value specifying how LineFeed characters will be handled in downloaded messages. Most Internet messages combine Carriage Return and Line Feed characters to indicate the end of a line. Macintosh applications prefer a carriage return only as the end-of-line character. This option lets users strip the linefeed character from their message text. A value of zero will leave the message "as is". A value of 1 will strip linefeed characters from messages. A value of -1 will leave this preference as it has been set previously. The default option defaults to 1 and will automatically strip any linefeeds found in messages.

msgFolder is a text value indicating the local pathname to a folder in which retrieved messages are stored by default.

Compatibility note (version 6.8.1): If the *MSG_SetPrefs* command is not used, the *stripLineFeed* and *msgFolder* parameters of the *POP3_SetPrefs* command will be taken into account if this command has been used previously. If the *MSG_SetPrefs* command is used, the *POP3_SetPrefs* parameters are ignored.

attachFolder is a text value containing the local pathname to a folder in which attachments are stored when the *MSG_Extract* command separates the attachments from the main body of a message.

Compatibility note (version 6.8.1): This parameter is also found in *POP3_SetPrefs* and *MSG_SetPrefs*; therefore, you can modify it using either of these two commands.
We strongly recommend that you use the *MSG_SetPrefs* command. The *POP3_SetPrefs* parameter is used for compatibility reasons and will no longer be used in the future. The *attachFolder* parameter of the *POP3_SetPrefs* command is optional; therefore, we recommend that you do not use this parameter. This recommendation also applies to *POP3_GetPrefs*.

IC File Transfer

File Transfer, Overview

The File Transfer Protocol (FTP) is the primary means of transferring documents and applications from one computer to another. FTP "sites" are computers throughout the world running FTP server software. The File Transfer Protocol provides a means for disparate systems to exchange files. Client applications on a variety of platforms can all log into a FTP server in order to upload or download text or binary files. The FTP routines within the 4D Internet Commands give developers the tools to create FTP clients within their 4D databases.

Notes:

- When specifying pathnames in the FTP commands, you should always treat file locations on the FTP site as a Unix directory, even if you know the FTP host to be a Macintosh running FTP server software. Whatever the platform, the FTP server software will internally convert your Unix pathname to the format it needs to serve its documents to connected clients.
- For greater flexibility, 4D Internet commands let you pass a POP3, IMAP or FTP connection reference directly to low-level TCP commands and vice versa. For more information, refer to the **Low Level Routines, Overview** section
- The FTP commands of 4D Internet Commands work with files whose names have a limited length. The following table provides the maximum number of characters based on the OS:

	Windows	**OS X**
32 bits	255	63
64 bits	255	1024

FTP_Append

FTP_Append (ftp_ID ; localPath ; hostPath ; progress) -> Function result

Parameter	Type		Description
ftp_ID	Longint	→	Reference to a FTP login
localPath	Text	→	Pathname of document to send
hostPath	Text	→	Pathname to destination of document
progress	Integer	→	1 = Show Progress, 0 = Hide Progress
Function result	Integer	↩	Error Code

Description

The *FTP_Append* command performs the same action as *FTP_Send* with the one exception that it will append the data being sent to the end of an existing file identified by the *hostPath* parameter. This command's primary function is to append data onto the end of pre-existing text files.

FTP_ChangeDir

FTP_ChangeDir (ftp_ID ; directory) -> Function result

Parameter	Type		Description
ftp_ID	Longint	→	Reference to a FTP login
directory	Text	→	Unix directory pathname
Function result	Integer	↩	Error Code

Description

The *FTP_ChangeDir* command changes your current working directory (CWD) to the path given to the *directory* parameter.

The commands *FTP_GetDirList* and *FTP_GetFileInfo* will also change the current working directory. However, executing the *FTP_ChangeDir* command is faster and needs less parameters.

ftp_ID is the long integer reference to the FTP session established with *FTP_Login*.

directory is a text value in the format of a HostPath which references an existing FTP directory. An error will be returned if the directory does not exist or if you do not have sufficient access priveleges to perform this action. In this case, the current working directory will be left unchanged.

Example

This statement will set the CWD to the FTP root:

```
$err:=FTP_ChangeDir(ftp_ID;"/")
```

FTP_Delete

FTP_Delete (ftp_ID ; hostPath) -> Function result

Parameter	Type		Description
ftp_ID	Longint	→	Reference to a FTP login
hostPath	Text	→	Pathname to document
Function result	Integer	↩	Error Code

Description

Given a pathname to a file in the format of a HostPath, the *FTP_Delete* command will delete the specified file from the remote FTP Server. An error will be returned if you do not have sufficient access priveleges to perform this action.

ftp_ID is the long integer reference to the FTP session established with *FTP_Login*.

hostPath is the text path to the document to be deleted. The value of the *hostPath* parameter may be a full pathname specification or simple file name. If the shortened form is used then the specified file must be within the CWD.

Note: You can change the CWD using the *FTP_ChangeDir* command. You can also know the CWD at any time using the *FTP_PrintDir* command.

FTP_GetDirList

FTP_GetDirList (ftp_ID ; directory ; names ; sizes ; kinds ; modDates ; modTimes) -> Function result

Parameter	Type		Description
ftp_ID	Longint	→	Reference to a FTP login
directory	Text	→	Unix directory pathname
		←	Current directory
names	String array	←	List of Names
sizes	Longint array	←	List of Sizes
kinds	Integer array	←	List of Kinds 0 = plain file, 1 = directory, 2 = block-type special file, 3 = character-type special file, 4 = symbolic link, 5 = FIFO special file, 6 = AF_UNIX address family socket
modDates	Date array	←	List of Modification Dates
modTimes	Longint array	←	List of Modification Times
Function result	Integer	↩	Error Code

Description

The *FTP_GetDirList* command will retrieve a list of the objects in a *directory* of the FTP session identified by *ftp_ID*. Information on the names, sizes, types, modification dates and, optionally, modification times of the *directory* items is returned in arrays. A connection to the FTP site must have already been opened via *FTP_Login* and still valid (*FTP_VerifyID*). The *FTP_GetDirList* command changes your current working directory (CWD) to the path given and returned to the *directory* parameter.

ftp_ID is the long integer reference to the FTP session established with *FTP_Login*.

directory is a text value in the format of a HostPath which references a FTP directory. A 4D variable or field must be passed to this parameter since the resulting "current directory" will be returned. Normally, the value returned to this parameter will be the same as the value passed to it. However, there may be cases (such as access restrictions) where the directory change was not successful. In this case, the *directory* parameter will hold the HostPath to the session's current directory.

A null string passed in this parameter will return the current directory file list into the arrays and the current directory's HostPath into the *directory* parameter.

names is a string or text array to hold the name of each object in the specified *directory*.

sizes is a long integer array to hold the sizes of the objects in *directory*.

kinds is an integer array to hold the type of each object in *directory*. The interpretation of this value is based on the following table:

Kind	File Type
0	plain file
1	directory
2	block-type special file
3	character-type special file
4	symbolic link (aliases on files or folders)
5	FIFO special file
6	AF_UNIX address family socket

Note: In the case of a symbolic link (kind=4), the FTP server returns a particular pathname (Alias name + symbol + pathname to the source file or folder). If you try to use this pathname to access source files or folders, an error will be returned. You MUST extract the pathname to the source file or folder from the string returned by *FTP_GetDirList* which starts just after the symbolic character. Otherwise, commands such as *FTP_GetFileInfo* will return the error –10085 since the file or folder will not be found.

modDates is a 4D date array to hold the last modified date for each object in *directory*.

modTimes is a longint array that receives the time of the last modification for each object in the *directory*.

Reminder: In 4D, longint arrays are used for handling time type data (each array item represents a number of seconds). Use the **Time string** command to convert these values into the **HH:MM:SS** format.

FTP_GetFileInfo

FTP_GetFileInfo (ftp_ID ; hostPath ; size ; modDate ; modTime) -> Function result

Parameter	Type		Description
ftp_ID	Longint	⇒	Reference to a FTP login
hostPath	Text	⇒	Pathname to document
size	Longint	⇐	Size of document
modDate	Date	⇐	Modification Date
modTime	Time	⇐	Modification Time
Function result	Integer	↩	Error Code

Description

The *FTP_GetFileInfo* command returns information concerning the last modification of the file designated by *hostPath.*

ftp_ID is the long integer reference to the FTP session established with *FTP_Login*.

hostPath is the text path to the document to return information about.

Note: The *FTP_GetFileInfo* command may modify the Current Working Directory (CWD) if *hostPath* is a full pathname which indicates a directory different from the CWD. In this case, the CWD becomes the directory defined by the *hostPath* parameter.

size is a long integer variable or field to hold the size of the file identified by *hostPath*.

modDate and *modTime* return the date and time of the last modification of the file.

FTP_GetPassive

FTP_GetPassive (ftp_ID ; passiveMode) -> Function result

Parameter	Type		Description
ftp_ID	Longint	→	Reference to a FTP login
passiveMode	Integer	←	Current data stream transfer mode 0=Active mode, 1=Passive mode
Function result	Integer	↩	Error Code

Description

The *FTP_GetPassive* command returns the current data stream transfer mode.
For more details about FTP transfer modes, refer to the *FTP_SetPassive* command description.

ftp_ID is the long integer reference to the FTP session established with *FTP_Login*.

passiveMode returns the current data stream transfer mode:

- if 0 is returned, the FTP Server is currently asked to work in Active mode.
- if 1 is returned, the FTP Server is currently asked to work in Passive mode (default value).

FTP_GetType

FTP_GetType (ftp_ID ; ftp_Mode) -> Function result

Parameter	Type		Description
ftp_ID	Longint	⇒	Reference to a FTP login
ftp_Mode	String	⇐	"A" = Ascii; "I" = Image; "L 8" = Logical 8-bit
Function result	Integer	↩	Error Code

Description

The *FTP_GetType* command returns information about the current FTP Transfer mode. The Transfer mode may be set using the *FTP_SetType* command.

ftp_ID is the long integer reference to the FTP session established with *FTP_Login*.

ftp_Mode returns a code describing the current FTP transfer mode.

FTP_Login

FTP_Login (hostName ; userName ; password ; ftp_ID ; welcomeText) -> Function result

Parameter	Type		Description
hostName	String	→	Host name or IP address
userName	String	→	User name
password	String	→	Password
ftp_ID	Longint	←	Reference to this new FTP session
welcomeText	Text	←	FTP Welcome text
Function result	Integer	↩	Error Code

Description

The *FTP_Login* command establishes a connection with the FTP server at *hostName* and logs onto the system using the supplied *userName* and **Password**.

hostName is the host name or IP address of the remote system.

userName is the name of a user account recognized by the FTP server. Many FTP servers support guest access via an "anonymous" username. If you are logging in anonymously, it is customary to supply your e-mail address as the password.

password is the password for *userName* on the system.

ftp_ID is the long integer value obtained for the newly opened session. This value will be used in subsequent FTP commands. This parameter must be passed a 4D variable or field in order to accept the returned results.

welcomeText is an optional parameter which contains the text returned when the user logs into the system. Many FTP sites have a Welcome message displayed at the time of login. If specified, this parameter must be passed a 4D variable or field in order to accept the returned results.

Example

```
$OK:=False
Case of
   :(FTP_Login("ftp.4d.com";"anonymous";"dbody@aol.com";vFTP_ID;vFTP_Msg)#0)
   :(FTP_Progress(-1;-1;"Progress window";"Getting requested file…";"*")#0)
   :(FTP_Send(vFTP_ID;"My Hard Drive:Documents ƒ:July Sales Report";"/pub/reports";1)#0)
   :(FTP_Logout(vFTP_ID)#0)
   Else
      $OK:=True `all commands executed without error
End case
```

Note: For more information about this particular use of the **Case of** structure, please refer to **Appendix A, Programming Tips**.

FTP_Logout

FTP_Logout (ftp_ID) -> Function result

Parameter	Type		Description
ftp_ID	Longint	⇒	Reference to a FTP login
		⇐	0 = session successfully closed
Function result	Integer	↩	Error Code

Description

Given a reference to an open FTP session, the *FTP_Logout* command will disconnect from the server and free any memory used by the session. This command will return the value of zero into the *ftp_ID* parameter upon successful close of the session.

ftp_ID is the long integer reference to the FTP session established with *FTP_Login*.

Example

```
 If(FTP_Login("ftp.4d.com";"anonymous";vEmailID;vFTP_ID;vFTP_Msg)=1)
    $error:=FTP_Send(vFTP_ID;"My Hard Drive:Documents:Sales Report";"/pub/reports";1)
    $error:=FTP_Logout(vFTP_ID)
 End if
```

FTP_MacBinary

FTP_MacBinary (ftp_ID ; macBinaryMode) -> Function result

Parameter	Type		Description
ftp_ID	Longint	→	Reference to a FTP login
macBinaryMode	Integer	→	-1 = Get Current setting, 1 = Enable, 0 = Disable
		←	Current setting (if -1 passed)
Function result	Integer	↩	Error Code

Description

The *FTP_MacBinary* command enables/disables the MacBinary mode setting for FTP transfers using *FTP_Send* and *FTP_Receive*. Given a current FTP session identified by *ftp_ID*, this command will either turn MacBinary transfers on or off depending on the value passed in the *macBinaryMode* parameter.

The MacBinary protocol is often used by Macintosh FTP clients and servers to facilitate the transfer of binary data or files that contain both data and resource forks.

Note for Windows users: It is possible to use the MacBinary protocol for FTP transfers in a Windows environment however it should be noted that it may often not make sense to decode a MacBinary file on a PC computer. Intel-based machines cannot store files containing both data and resource forks. Since such a file format is foreign to the PC platform, Macintosh files which contain a resource fork are likely to be corrupted if saved in an unencoded format.

ftp_ID is the long integer reference to the FTP session established with *FTP_Login*.

macBinaryMode is integer parameter indicating whether to turn MacBinary transfers on or off. This value should be passed as a variable so the command can return the state of MacBinary transfers after the attempted change. Passing a 1 will enable MacBinary and a zero will disable. A -1 value in the parameter will cause the command to return in the *macBinaryMode* parameter the current state of MacBinary transfers (1 or zero).

Warning: Not all FTP servers support the MacBinary protocol, in this case the error 10053 is returned at each *FTP_MacBinary* command call, whatever the value of the *macBinaryMode* parameter. Previously described behaviours become false.

Example

This example enables the MacBinary protocol before receiving an FTP file. If the file was successfully received with MacBinary turned on then it is decoded into its original format and the MacBinary document is deleted.

```
vUseMacBin:=-1
$error:=FTP_MacBinary(vFTP_ID;vUseMacBin)
If($error=10053)
   MacBinaryIsSupported:=False `Ftp server doesn't support the MacBinary protocol
Else
   MacBinaryIsSupported:=True
End if


vLocalFile:=""
If(MacBinaryIsSupported)
   vUseMacBin:=1
   $error:=FTP_MacBinary(vFTP_ID;vUseMacBin) `Try to turn MacBinary on for the download
End if
$error:=FTP_Receive(vFTP_ID;"MyApplication";vLocalFile;cbShowTherm)
If($error=0)&(vUseMacBin=1) `If received OK and the file is in MacBinary format
   vDecodePath:=""
   If(IT_Decode(vLocalFile;vDecodePath;8)=0) `MacBinary decode
      DELETE DOCUMENT(vLocalFile) `If sucessful decode of source, then delete it.
   End if
End if
```

FTP_MakeDir

FTP_MakeDir (ftp_ID ; directory) -> Function result

Parameter	Type		Description
ftp_ID	Longint	⇒	Reference to a FTP login
directory	Text	⇒	Unix directory pathname
Function result	Integer	⇐	Error Code

Description

Given an acceptable *directory* name, the *FTP_MakeDir* command will create a new Folder directory within the Current Working Directory (CWD). An error will be returned if you do not have sufficient access privileges to perform this action.

ftp_ID is the long integer reference to the FTP session established with *FTP_Login*.

directory is a text value in the format of a HostPath which references a FTP directory. The value of the *directory* parameter may be a full pathname specification or simple folder name. If the shortened form is used then the directory is created within the CWD. It is recommended that the *directory* name **not** contain any blank spaces.

Note: You can change the CWD using the *FTP_ChangeDir* command. You can also determine the CWD at any time using the *FTP_PrintDir* command.

FTP_PrintDir

FTP_PrintDir (ftp_ID ; directory) -> Function result

Parameter	Type		Description
ftp_ID	Longint	→	Reference to a FTP login
directory	Text	←	Unix directory pathname
Function result	Integer	↩	Error Code

Description

The *FTP_PrintDir* command returns your current working directory (CWD) in the *directory* parameter.

The command *FTP_GetDirList* will also return the current working directory. However, executing the *FTP_PrintDir* command is faster and needs less parameters.

ftp_ID is the long integer reference to the FTP session established with *FTP_Login*.

The *directory* parameter returns the current working directory (CWD).

Example

This example will return the current working directory in the $Cwd variable.

```
$err:=FTP_PrintDir(ftp_ID;$Cwd)
```

FTP_Progress

FTP_Progress (left ; top ; windowTitle ; thermoText ; cancel) -> Function result

Parameter	Type		Description
left	Integer	→	Left window coordinate
top	Integer	→	Top window coordinate
windowTitle	String	→	Thermometer Window Title
thermoText	String	→	Text above thermometer progress
cancel	String	→	Cancel button text
Function result	Integer	↩	Error Code

Note concerning 64-bit version for OS X

This command is not supported with the 64-bit version of 4D Internet Commands for OS X. When it is called from this platform, it returns the error -2201 (*unimplemented feature*).

Description

The *FTP_Progress* command defines window coordinates and dialog box text for the FTP progress indicator. The progress indicator can appear during calls to *FTP_Send*, *FTP_Append* or *FTP_Receive*. The *FTP_Progress* command does not display the progress window itself, it only defines the windows characteristics for when it is displayed by the send and receive commands. Both *FTP_Send*, *FTP_Append* and *FTP_Receive* have parameters which can show or hide the progress window.

The progress window will automatically close upon completion of a file transfer. If for some reason the size of the file being sent or received cannot be determined, the thermometer will be displayed as a barber pole and the file size will be displayed as "unknown".

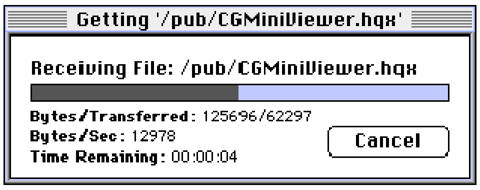
left is the coordinates of the left side of the thermometer progress window. If *left* is -1, the window will be centered horizontally on the screen.

top is the coordinates of the top side of the thermometer progress window. If *top* is -1, the window will be centered vertically on the screen.

windowTitle is the title of the thermometer progress window. In the following example, the window title is "Getting '/pub/CGMiniViewer.hqx'" If *windowTitle* is a null string, the window will have no title.

thermoText is the text to be displayed above the progress thermometer. If *thermoText* is "*" then the text will be the default. The default text for the thermometer is the status text of the transfer process, sent by the host. This text changes as the connection goes through the different stages of the transfer process.

cancel is the text of the **Cancel** button. If *cancel* is a null string, the **Cancel** button will be hidden. If *cancel* is "*", the text will be the default text, which is "Cancel".


Getting '/pub/CGMiniViewer.hqx'

Receiving File: /pub/CGMiniViewer.hqx

Bytes/Transferred: 125696/62297
Bytes/Sec: 12978
Time Remaining: 00:00:04

Example

```
$error:=FTP_Progress(-1;-1;"Getting '/pub/CGMiniViewer.hqx'";"*";"*")
Case of
   :(FTP_Login("ftp.4d.com";"anonymous";"dbody@aol.com";vFTP_ID;vFTP_Msg)#0)
   :(FTP_Receive(vFTP_ID;"/pub/CGMiniViewer.hqx";"HardDrive:Docsf:4D";1)#0)
   :(FTP_Logout(vFTP_ID)#0)
   Else
      $OK:=True `all commands executed without error
End case
```

Note: For more information about this particular use of the **Case of** structure, please refer to **Appendix A, Programming Tips**.

FTP_Receive

FTP_Receive (ftp_ID ; hostPath ; localPath ; progress) -> Function result

Parameter	Type		Description
ftp_ID	Longint	→	Reference to a FTP login
hostPath	Text	→	Pathname of document to receive
localPath	Text	→	Pathname to destination of document
		←	Resulting file pathname (if "" passed)
progress	Integer	→	0 = Hide Progress, 1 = Show Progress
Function result	Integer	↩	Error Code

Description

The *FTP_Receive* command receives a file using the File Transfer Protocol from the path referenced by *hostPath*. *FTP_Receive* will return an error# -48 if the destination file already exists.

ftp_ID is the long integer reference to the FTP session established with *FTP_Login*.

hostPath is a text value that specifies the path of the document to be received. If *hostPath* is not a full path to a document, the command will return an error. As with all paths to Unix documents, the path should be separated by slashes ("/"). For more information, see the section entitled **Glossary and Terminology** at the beginning of the manual.

localPath is a text value that specifies the path of the destination of the document. If *localPath* is a null string, the user will be presented with a Standard Save-File Dialog and the resulting file pathname will be returned back into the *localPath* variable. If *localPath* contains only a filename, the file will be saved in the same folder as the structure of the database (with 4D single-user) or the 4D Client folder (with 4D Server). As with all paths to local documents, the path should be separated by the delimiter appropriate for the platform the externals are being used. For more information, see the section entitled **Glossary and Terminology** at the beginning of the manual.

progress is an integer value indicating whether the Progress indicator should be displayed or not. A value of 1 will display the progress indicator. A value of zero will hide the progress indicator.

Example

```
vUseMacBin:=-1
$error:=FTP_MacBinary(vFTP_ID;vUseMacBin)
If($error=10053)
   MacBinaryIsSupported:=False `Ftp server doesn't support the MacBinary protocol
Else
   MacBinaryIsSupported:=True
End if

vLocalFile:=""
If(MacBinaryIsSupported)
   vUseMacBin:=1
   $error:=FTP_MacBinary(vFTP_ID;vUseMacBin) `Try to turn MacBinary on for the download
   $error:=FTP_Receive(vFTP_ID;"CGMiniViewer.hqx";vLocalFile;cbShowTherm)
   If($error=0)&(vUseMacBin=1)
      vDecodePath:=""
      If(IT_Decode(vLocalFile;vDecodePath;8)=0) `MacBinary decode
         DELETE DOCUMENT(vLocalFile) `If successful decode of source, then delete it.
      End if
   End if
End if
```

FTP_RemoveDir

FTP_RemoveDir (ftp_ID ; directory) -> Function result

Parameter	Type		Description
ftp_ID	Longint	⇨	Reference to a FTP login
directory	Text	⇨	Unix directory pathname
Function result	Integer	↩	Error Code

Description

Given an acceptable *directory* name, the *FTP_RemoveDir* command will delete a Folder directory. An error will be returned if you do not have sufficient access privileges to perform this action. Additionally, attempting to remove a directory which contains items will likely result in a security error.

ftp_ID is the long integer reference to the FTP session established with *FTP_Login*.

directory is a text value in the format of a HostPath which references an existing FTP directory. The value of the *directory* parameter may be a full pathname specification or simple folder name. If the shortened form is used then the specified directory must be within the CWD.

Note: You can change the CWD using the *FTP_ChangeDir* command. You can also know the CWD at any time using the *FTP_PrintDir* command.

FTP_Rename

FTP_Rename (ftp_ID ; hostPath ; newName) -> Function result

Parameter	Type		Description
ftp_ID	Longint	→	Reference to a FTP login
hostPath	Text	→	Pathname to document on FTP Server
newName	Text	→	New document name
Function result	Integer	↩	Error Code

Description

Given a pathname to a file in the format of a *hostPath*, the *FTP_Rename* command will rename the specified file on the remote FTP Server. An error will be returned if you do not have sufficient access priveleges to perform this action.

ftp_ID is the long integer reference to the FTP session established with *FTP_Login*.

hostPath is the text path to the document to be renamed. The value of the *hostPath* parameter may be a full pathname specification or simple file name. If the shortened form is used then the specified file must be within the CWD.

newPathName contains the value you wish to rename the remote document.

Note: You can change the CWD using the *FTP_ChangeDir* command. You can also know the CWD at any time using the *FTP_PrintDir* command.

FTP_Send

FTP_Send (ftp_ID ; localPath ; hostPath ; progress) -> Function result

Parameter	Type		Description
ftp_ID	Longint	⇒	Reference to a FTP login
localPath	Text	⇒	Pathname of document to send
hostPath	Text	⇒	Pathname to destination of document
progress	Integer	⇒	1 = Show Progress, 0 = Hide Progress
Function result	Integer	↩	Error Code

Description

Given a reference to an open FTP session, the pathname of a document to send and the destination pathname, the *FTP_Send* command sends the document to the remote machine. *FTP_Send* will return immediately if a FTP file status error occurs.

ftp_ID is the long integer reference to the FTP session established with *FTP_Login*.

localPath is the path of the document to be sent. If *localPath* is a null string, the user will be presented with the Standard Open File dialog. If *localPath* is a file name with no path, the command will look in the folder that contains the database structure (with 4D single-user) or in the 4D Client folder (with 4D Server) for the file. As with all paths to local documents, the directories should be seperated by a delimiter appropriate for the platform. For more information, see the section entitled **Glossary and Terminology** at the beginning of the manual.

Note: FTP commands work with documents whose names have a limited length. For more information, refer to the **File Transfer, Overview** section).

hostPath is the path to the destination of the document, including the file name. The *hostPath* represents the desired name of the file once it has been received by the FTP server. If *localPath* is a null string allowing the user to pick a file from disk, then *hostPath* may also be a null string, in which case the chosen file's name will be used.

hostPath may be either a full pathname specification or simply a filename. If a full pathname is supplied, the specified file will be placed in the directory indicated by *hostPath*. If only a filename is provided, or null strings are used in the file selection, then the file will be sent to the Current Working Directory [CWD].

If the file or pathname cannot be resolved correctly, the command will return an error. If the user does not have enough privileges to send a file to that directory, an error will be returned. As with all paths to Unix documents, the path should be separated by slashes ("/"). For more information, see the section entitled **Glossary and Terminology** at the beginning of the manual.

Note: The FTP server may also impose specific limitations concerning the length or characters used for file names.

progress is an integer value indicating whether the Progress indicator should be displayed or not. A value of 1 will display the progress indicator. A value of zero will hide the progress indicator.

Example 1

```
 $OK:=False
 Case of
    :(FTP_Login("ftp.4d.com";"anonymous";vEmailID;vFTP_ID;vFTP_Msg)#0)
    :(FTP_Progress(-1;-1;"Progress window";"Getting requested file…";"Cancel")#0)
    :(FTP_Send(vFTP_ID;"My Hard Drive:Documents:July Sales
Report";"/pub/reports/July_sales";1)#0)
    :(FTP_Logout(vFTP_ID)#0)
    Else
       $OK:=True `all commands executed without error
 End case
```

Note: For more information about this particular use of the **Case of** structure, please refer to **Appendix A, Programming Tips**.

Example 2

```
$error:=FTP_Send(vFTP_ID;"";"";1)
```

FTP_SetPassive

FTP_SetPassive (ftpID ; passiveMode) -> Function result

Parameter	Type		Description
ftpID	Longint	→	Reference to a FTP login
passiveMode	Integer	→	0=Active mode, 1=Passive mode (default mode)
Function result	Integer	↩	Error Code

Description

The *FTP_SetPassive* command sets the data stream transfer mode between an FTP Server and an FTP Client while executing commands such as *FTP_GetDirList*, *FTP_Send*, *FTP_Append* or *FTP_Receive*. Data stream transfer mode setting will be applied to these commands once the command *FTP_SetPassive* has been executed.

Exchanges between a FTP Server and a FTP Client are based upon two streams: the control stream (port 21 by default) and the data stream (port 20 by default). Usually, FTP Servers are defined as "active" since they open and manage the data connection.
For historical reasons, 4D Internet Commands default data stream transfer mode consists of asking the FTP Server to work in Passive mode. It means that prior to each exchange on the data stream, the FTP command "PASV" is sent.
However, some FTP Servers do not support the passive mode, and also firewalls may not allow it. In these cases, you will need to set the active mode as the current data stream transfer mode.

Note: You may need to check with the network administrator whether the active or the passive mode is used for FTP exchanges.

ftp_ID is the long integer reference to the FTP session established with *FTP_Login*.

The *passiveMode* parameter value specifies the data stream transfer mode:

- a value of 0 will specify the FTP Server to work in Active mode
- a value of 1 will specify the FTP Server to work in Passive mode (default value).

FTP_SetType

FTP_SetType (ftp_ID ; ftp_Mode) -> Function result

Parameter	Type		Description
ftp_ID	Longint	→	Reference to a FTP login
ftp_Mode	String	→	"A" = Ascii; "I" = [Default] Image; "L 8" = Logical 8-bit
Function result	Integer	↩	Error Code

Description

The *FTP_SetType* command is used to alter the FTP transfer mode used during Send/Receive operations. Typically this will not need to be changed from the default settings. However, because of differences between various platforms and ftp implementations, it may be necessary to change the mode for certain types of FTP transfers. In particular, some transfers of plain-text documents may require you to switch the mode to Ascii in order to properly transfer the text file.

ftp_ID is the long integer reference to the FTP session established with *FTP_Login*.

ftp_Mode should contain a code as described above indicating the preferred transfer mode to use for future Send/Receive operations. By default, the Image ("I") transfer mode is used.

FTP_System

FTP_System (ftp_ID ; systemInfo) -> Function result

Parameter	Type		Description
ftp_ID	Longint	⇨	Reference to a FTP login
systemInfo	String	⇦	System Information
Function result	Integer	↩	Error Code

Description

The *FTP_System* command simply obtains information into *systemInfo* that describes the FTP server software.

ftp_ID is the long integer reference to the FTP session established with *FTP_Login*.

systemInfo will contain information about the FTP server.

FTP_VerifyID

FTP_VerifyID (ftp_ID) -> Function result

Parameter	Type		Description
ftp_ID	Longint	→	Reference to a FTP login
		←	0 = Connection has already closed
Function result	Integer	↩	Error Code

Description

A FTP server will disconnect accounts which do not show activity in a period of time determined by its administrator. Each command that interacts with the FTP server will force a reset of your inactivity timer. The *FTP_VerifyID* command resets the inactivity time for the specified FTP session without altering the current state or directory. This allows the user to keep a session active if the possibility exists that the session may timeout.

When executed, the *FTP_VerifyID* command will verify that the connection has not already been closed. If the session is still open the command will tell the FTP server to reset the timeout counter for the session back to zero. If the connection has already closed, *FTP_VerifyID* will return the appropriate error, free memory used by the FTP session, and return a zero value back to *ftp_ID*.

ftp_ID is the long integer reference to the FTP session established with *FTP_Login*.

IC IMAP Review Mail

IMAP4 Commands, Overview

The set of IMAP commands enables your database to access and manipulate electronic mail messages over an IMAP electronic mail server and to retrieve electronic messages from your IMAP server. IMAP commands are compliant with the Internet Message Access Protocol, Version 4 revision 1 (IMAP4rev1), defined by rfc 2060. IMAP4rev1 allows the managing of remote message folders, called "mailboxes", in a way that is functionally similar to local mailboxes.

IMAP commands include operations to create, delete and rename mailboxes; check for new messages; permanently remove messages; set and clear message flags; search messages; and retrieve selective message parts.

Terminology

"Connection" refers to the entire sequence of IMAP client/server interaction from the initial network connection (*IMAP_Login*) until the end of the selection (*IMAP_Logout*).

"Session" refers to the sequence of client/server interaction from the moment a mailbox is selected (*IMAP_SetCurrentMB*) until the end of the selection (*IMAP_SetCurrentMB*, *IMAP_CloseCurrentMB*) or until the connection is closed (*IMAP_Logout*).

IMAP Connection Overview

- Initializing TCP communication: *IT_MacTCPInit* (the *IT_PPPConnect* command must be called before *IT_MacTCPInit* in the event of a PPP connection).
- Opening a connection: *IMAP_Login*
- Managing mailboxes: List, create, delete, rename, subscribe/unsubscribe, and/or get status parameters.
- Opening a session by defining the current working mailbox: *IMAP_SetCurrentMB*.
  Once the current mailbox is set, you can manage messages for it.
- Managing messages: List, download or delete messages; list message flags; modify message flags; copy to another mailbox; search and retrieve e-mail parts without any downloading, etc.
- Once you are finished working with the current mailbox messages, you can close the session or open a new one by setting another current mailbox. In any case, the IMAP server will permanently update its messages. For instance, it will delete all messages with the **¥Deleted** flag set.
- Once you are finished, you should logout. Closing a connection: *IMAP_Logout*.
- Other operations: Preference settings, capability, check connection, and reset any inactivity auto-logout timer on the IMAP server.

IMAP Command Themes

The IMAP-related commands are divided into two sections: **IC IMAP Review Mail** and **IC Downloaded Mail**. These commands have been separated to show the different methods of reading electronic mail. When reading electronic mail from an IMAP server, messages (or message information) may be imported into 4D structures (variables, fields, arrays) or downloaded to the disk. This section details the 4D Internet Commands' capacity to read messages from an IMAP server.

The need for dual message-retrieval methods is due mainly to memory constraints for actions that may download large amounts of information. For instance, a single message containing a 5-MB attachment could easily overflow the database's storage capacity. A picture or BLOB field is the only 4D structure capable of storing something of this size; however, converting a message or attachment to this format is not very efficient since accessing the picture or BLOB involves cumbersome memory requirements for the client. To resolve this problem, this section has an *IMAP_Download* command which transfers messages from the IMAP server to the user's hard disk.

Once imported to the disk, the "**IC Downloaded Mail**" section details the commands used to manipulate local files.

Mailbox mechanisms

An IMAP mailbox can be handled like a folder and may contain files and/or subfolders. Similarly, a mailbox may contain messages and/or submailboxes.

A mailbox is accessed using its complete hierarchical name. Depending on the IMAP server, each hierarchical level is separated by a hierarchy separator (a separator is returned using the *IMAP_ListMBs* command).

You can use the separator to create child mailboxes and to search higher or lower levels of the naming hierarchy. All children of a top-level hierarchy node use the same separator character.

Note: Messages can only be managed once the current working mailbox has been selected (*IMAP_SetCurrentMB*).

Each account can have one or several mailboxes.

Mailbox names are still case-sensitive; therefore, you cannot create two mailboxes with names that differ only in case.

The INBOX mailbox is a particular case: it exists in every account and is used for storing incoming messages. The INBOX is created automatically whenever an account is set up.

A user cannot remove the INBOX mailbox but may rename it. If he chooses to rename it, a new empty INBOX is immediately created. The INBOX name is never case-sensitive.

Some mailbox attributes, such as the total number of messages or new messages, may be checked even if the mailbox is not the current one.

msgNum and uniqueID

When using IMAP commands, it is important to fully understand the most frequently used parameters; more particularly, mailbox mechanisms *msgNum* and *uniqueID*.

msgNum is the number of a message in the mailbox at the time the *IMAP_SetCurrentMB* command is executed.

Once a current mailbox is selected, messages in the mailbox are assigned numbers starting from 1 up to the total number of items in the mailbox. Numbers are assigned based on the order that the messages were received in the mailbox with 1 being the oldest. The numbers assigned to messages are only valid from the moment you select the current working mailbox (*IMAP_SetCurrentMB*) until it is closed (*IMAP_CloseCurrentMB*, *IMAP_SetCurrentMB* or *IMAP_Logout*).

When the mailbox is closed, all messages marked for deletion will be removed.

When the user logs back onto the IMAP server, the current messages in the mailbox will once again be numbered from 1 to X. For instance, if there are 10 messages in the mailbox, and messages 1 to 5 are deleted, when the user reopens the mailbox, the former messages 6 to 10 will have been re-numbered 1 to 5.

For instance, consider the example below: You log on to an IMAP server and obtain the following list of messages:

msgNum	uniqueID	Date	From	Subject
1	10005	1 Jul 2001 ...	danw@acme.com	Sales lead...
2	10008	1 Jul 2001 ...	frank@acme.com	Site-License order
3	10012	3 Jul 2001 ...	joe@acme.com	Lunch anyone?
4	20000	4 Jul 2002 ...	kelly@acme.com	Your wife called...
5	20001	4 Jul 2002 ...	track@fedex.com	FedEx tracking

During this session, you delete messages number 3 and 4. When you close the current working mailbox, the requested deletions are made. When you log back onto the server, your message list will be re-numbered as follows:

msgNum	uniqueID	Date	From	Subject
1	10005	1 Jul 2001 ...	danw@acme.com	Sales lead...
2	10008	1 Jul 2001 ...	frank@acme.com	Site-License order
3	20001	4 Jul 2002 ...	track@fedex.com	FedEx tracking

msgNum is not a static value and will vary from one session to another. It will change in relation to other messages in the mailbox at the time the current working mailbox is selected.

However , the *uniqueID* is a unique number, assigned to the message by the IMAP server in a strictly ascending order. As each message is added to the mailbox, it is assigned a higher ID than the previously added message(s).

Unfortunately, IMAP servers do not use the *uniqueID* as the primary reference for their messages. When using

IMAP commands you will need to specify the *msgNum* as the reference for messages on the server. Developers may need to be careful when developing solutions which import message references into a database, while leaving the message body itself on the server.

Recommendations

Since the whole point of IMAP is interoperability, and since the latter cannot be tested in a vacuum, the final recommendation is “Test for EVERYTHING.” Therefore, test your client against every server you can get an account on.

For more information, please check out the following sites:

- IMAP Products and Services: *http://www.imap.org/products.html*
- MailConnect: *http://www.imc.org/imc-mailconnect*.

POP3 and IMAP4 commands comparison

Login	Exactly equiv	No POP parameter for IMAP
VerifyID	Exactly equiv	
Delete	Exactly equiv	IMAP commands delete in real time. POP3 waits for POP3_Logout to remove messages permanently. IMAP_SetFlags with ¥Deleted flag allows you to obtain the same result as the POP3_Delete command
Logout	Exactly equiv	
SetPrefs	Exactly equiv	No attachFolder for IMAP, POP3 attachFolder has become optional
GetPrefs	Exactly equiv	See attachFolder note in SetPrefs
MsgLstInfo	Exactly equiv	
MsgInfo	Exactly equiv	
MsgLst	Exactly equiv	
UIDToMsgNum	Exactly equiv	IMAP msgUID is a Longint, POP3 msgUID is a string
Download	Exactly equiv	
POP3_Reset	No direct equiv	Need combination of IMAP_Search on ¥Deleted flags and IMAP_SetFlags to remove the ¥Deleted flag
POP3_BoxInfo	No direct equiv	Need combination of IMAP_SetCurrentMB &IMAP_MsgLstInfo commands
IMAP_ MsgNumToUID	No direct equiv	
GetMessage	Almost Equiv	IMAP is more powerful since it allows you to choose one additional msgPart which is "only body"
POP3_Charset	No Equiv	IMAP automatically manages charset
IMAP_Capability	No Equiv	Specific to IMAP protocol
IMAP_ListMBs	No Equiv	Specific to IMAP protocol
IMAP_GetMBStatus	No Equiv	Specific to IMAP protocol
IMAP_SetCurrentMB	No Equiv	Specific to IMAP protocol
IMAP_GetCurrentMB	No Equiv	Specific to IMAP protocol
IMAP_CloseCurrentMB	No Equiv	Specific to IMAP protocol
IMAP_CopyToMB	No Equiv	Specific to IMAP protocol
IMAP_SubscribeMB	No Equiv	Specific to IMAP protocol
IMAP_CreateMB	No	Specific to IMAP protocol

IMAP_CreateMB	No Equiv	Specific to IMAP protocol
IMAP_DeleteMB	No Equiv	Specific to IMAP protocol
IMAP_RenameMB	No Equiv	Specific to IMAP protocol
IMAP_SetFlags	No Equiv	Specific to IMAP protocol
IMAP_GetFlags	No Equiv	Specific to IMAP protocol
IMAP_Search	No Equiv	Specific to IMAP protocol
IMAP_MsgFetch	No Equiv	Specific to IMAP protocol

Notes:

- IMAP and POP3 servers: in the case of the IMAP server, do not type *msgID* the same way since *msgID* is a Long Integer.
- Deletion does not work in exactly the same way between POP3 and IMAP protocols. *IMAP_Delete* removes messages in real time. To get the same result as *POP3_Delete*, use the *IMAP_SetFlags* to set the **¥Deleted** flag; to get the same result as *POP3_Reset*, use the *IMAP_SetFlags* to retrieve the **¥Deleted** flags.
- For greater flexibility, 4D Internet commands let you pass a POP3, IMAP or FTP connection reference directly to low-level TCP commands and vice versa. For more information, refer to the **Low Level Routines, Overview** section

IMAP_Capability

IMAP_Capability (imap_ID ; capability) -> Function result

Parameter	Type		Description
imap_ID	Longint	⇒	Reference to this IMAP login
capability	Text	⇐	IMAP capabilities
Function result	Integer	↩	Error code

Description

The *IMAP_Capability* command returns a text area containing a space-separated listing of capability names supported by the IMAP server. This list determines what version of IMAP, and what optional features (such as extension, revision or amendment to the IMAP4rev1 protocol), a server supports.

IMAP4rev1 must appear in the capability text to ensure compliance with 4D Internet Commands.

IMAP_CloseCurrentMB

IMAP_CloseCurrentMB (imap_ID) -> Function result

Parameter	Type		Description
imap_ID	Longint	→	Reference to an IMAP login
Function result	Integer	↩	Error code

Description

The *IMAP_CloseCurrentMB* command closes the current working mailbox without selecting another mailbox or executing an *IMAP_Logout*. *IMAP_CloseCurrentMB* permanently removes all messages that have the **¥Deleted** flag set.

Note: IMAP allows users to work concurrently with the same mailbox in a client/server mode. Suppose that someone carries out synchronization and keeps the connection open, the last mailbox used will remain in selected mode. Anyone else who tries to use this mailbox will not have valid information, or will not be able to work properly, depending on the server implementation, even if the user works in “disconnected mode” (i.e. connected but working using data).

imap_ID is a long integer reference to an open connection created with *IMAP_Login*.

IMAP_CopyToMB

IMAP_CopyToMB (imap_ID ; startMsg ; endMsg ; mbNameTarget ; msgDelete) -> Function result

Parameter	Type		Description
imap_ID	Longint	⇒	Reference to an IMAP login
startMsg	Longint	⇒	Start message number
endMsg	Longint	⇒	End message number
mbNameTarget	Text	⇒	Name of the destination mailbox
msgDelete	Integer	⇒	0= Do not remove from source mailbox, 1= Remove from source mailbox
Function result	Integer	⇐	Error code

Description

Given a range of messages from *startMsg* to *endMsg*, the *IMAP_CopyToMB* command will copy the specified message(s) to the end of the specified *mbNameTarget* destination mailbox. The flags and the internal date of the message(s) are usually preserved in the destination mailbox, depending on the IMAP server implementation.

After being copied, original messages are not removed from the source mailbox. If you want to remove them, you can use one of the 3 following processes:

- use the *IMAP_Delete* command,
- set the *msgDelete* optional parameter to 1,
- set *IMAP_SetFlags* (**¥Deleted**): the messages will be removed when the session is closed.

Note: The *msgDelete* parameter will force the execution of an *IMAP_Delete*; therefore, the deletion will include messages between *startMsg* and *endMsg* and ALL messages for which the **¥Deleted** flag is set.

If the destination mailbox does not exist, an error is returned.

imap_ID is a long integer reference to an open connection created with *IMAP_Login*.

startMsg is a long integer number that specifies the starting message number of the message range to be copied. The message number is a value representing the position of a message within the list of all messages in the mailbox identified by *imap_ID*.

endMsg is a long integer number that specifies the ending message number of the message range to be copied. The message number is a value representing the position of a message within the list of all messages in the mailbox identified by *imap_ID*.

Note: The *IMAP_Delete*, *IMAP_MsgLstInfo*, *IMAP_MsgLst*, *IMAP_SetFlags*, *IMAP_GetFlags* and *IMAP_CopyToMB* commands do not return an error if the *startMsg* is greater than the *endMsg*. In the event that this occurs, the command – in effect – does nothing.

mbNameTarget is the full name of the mailbox where the specified messages will be copied.

The *msgDelete* optional parameter allows setting if you want to remove the message from the source mailbox.
• 0= Do not remove from source mailbox (default value);
• 1= Remove from source mailbox.

If *msgDelete* is omitted, the default value is used.

If copying fails, the message is not removed from the source mailbox.

If the user does not have access to remove messages, an error message is generated.

IMAP_CreateMB

IMAP_CreateMB (imap_ID ; mbName) -> Function result

Parameter	Type		Description
imap_ID	Longint	→	Reference to an IMAP login
mbName	Text	→	Name of the mailbox to be created
Function result	Integer	↩	Error code

Description

The *IMAP_CreateMB* command creates a mailbox with the given name. If the IMAP server's hierarchy separator character appears elsewhere in the mailbox name, the IMAP server will create any parent names needed to create the given mailbox.

In other words, an attempt to create "Projects/IMAP/Doc" on a server in which "/" is the hierarchy separator character will create:

- Only the "Doc" mailbox if "Projects" & "IMAP" already exist.
- "IMAP" & "Doc" mailboxes if only "Projects" already exists.
- "Projects" & "IMAP" & "Doc" mailboxes, if they do not already exist.

imap_ID is a long integer reference to an open connection created with *IMAP_Login*.

mbName is the full name of the mailbox to be created (see naming rules in the IMAP introduction).

Note: Attempting to create an INBOX (which is a special name reserved to mean "the primary mailbox for this user on this server") or a mailbox with a name referring to an existing mailbox will lead to an error.

IMAP_Delete

IMAP_Delete (imap_ID ; startMsg ; endMsg) -> Function result

Parameter	Type		Description
imap_ID	Longint	→	Reference to an IMAP login
startMsg	Longint	→	Start message number
endMsg	Longint	→	End message number
Function result	Integer	↩	Error code

Description

The *IMAP_Delete* command sets the **¥Deleted** flag for the *startMsg* to *endMsg* range of messages and then deletes all messages where the **¥Deleted** flag is set (including messages where the **¥Deleted** flag has previously been set for the current session). Deletion is executed by the IMAP server and takes place when closing the connection (*IMAP_Logout*) or selecting another current mailbox (*IMAP_SetCurrentMB*) or closing the current mailbox (*IMAP_CloseCurrentMB*).

If you do not want to delete right away, you can use the *IMAP_SetFlags* command and set the **¥Deleted** flag to delete messages later.

imap_ID is a long integer reference to an open connection created with *IMAP_Login*.

startMsg is a long integer number which is the starting message number of the messages to delete.

endMsg is a long integer number which is the ending message number of the messages to delete.

Note: The *IMAP_Delete*, *IMAP_MsgLstInfo*, *IMAP_MsgLst*, *IMAP_SetFlags*, *IMAP_GetFlags* and *IMAP_CopyToMB* commands do not return an error if the *startMsg* is greater than the *endMsg*. In the event that this occurs, the command – in effect – does nothing.

IMAP_DeleteMB

IMAP_DeleteMB (imap_ID ; mbName) -> Function result

Parameter	Type		Description
imap_ID	Longint	⇒	Reference to an IMAP login
mbName	Text	⇒	Mailbox name to be deleted
Function result	Integer	↩	Error code

Description

The *IMAP_DeleteMB* command permanently removes the mailbox of a given name. Attempting to delete an INBOX or a mailbox that does not exist will generate an error.

The *IMAP_DeleteMB* command cannot delete a mailbox which has child mailboxes and which also has the **¥Noselect** mailbox attribute.

It is possible to delete a mailbox that has child names and does not have the **¥Noselect** mailbox attribute. In this case, all messages in the mailbox are removed and it acquires the **¥Noselect** mailbox attribute.

Note: The IMAP protocol does not guarantee that you can delete a mailbox which is not empty, though on some servers this is allowed. If you do choose to attempt it, you must be prepared to use another method should the more convenient one fail. Further, you should not try to delete the current working mailbox while it is open, but should first close it; some servers do not permit deletion of the current mailbox.

imap_ID is a long integer reference to an open connection created with *IMAP_Login*.

mbName is the full name of the mailbox to be deleted.

IMAP_Download

IMAP_Download (imap_ID ; msgNum ; headerOnly ; fileName ; updateSeen) -> Function result

Parameter	Type		Description
imap_ID	Longint	→	Reference to an IMAP login
msgNum	Longint	→	Message number
headerOnly	Integer	→	0 = Entire message, 1 = Only header
fileName	Text	→	Local Filename
		←	Resulting Local Filename
updateSeen	Integer	→	0 = Add ¥Seen Flag; 1= Do not add ¥Seen Flag
Function result	Integer	↩	Error code

Description

The *IMAP_Download* command is designed to retrieve a message from an IMAP server by downloading it to a disk-based file. Any IMAP message which contains attachments or whose size is greater than 32K should be downloaded with this command. File attachments can only be extracted from the messages retrieved in this way.

imap_ID is a long integer reference to an open connection created with *IMAP_Login*.

msgNum is a long integer value indicating which message in the mailbox to retrieve. *msgNum* represents the position of a message within the current list of messages. You cannot rely on the *msgNum* remaining the same for a specific e-mail item from session to session.

headerOnly is an integer value which denotes whether to retrieve the entire contents of the message or just the header information.

fileName contains the name of the file and the optional path where you would like the message saved. This value may be specified in three different ways:

- "" = Saves the file in the folder set by *IMAP_SetPrefs*, with the name “temp1” (if a file with the same name already exists, the filenames “temp2”, “temp3”, etc. will be tried until an unused file name is found).
- "FileName" = Saves the file in the folder set by *IMAP_SetPrefs* entitled FileName.
- "Path:FileName" = Saves the file in the path specified with the name FileName.

In the first two cases, if no folder has been specified by *IMAP_SetPrefs*, the message will be saved in the same folder as the structure of the database (with 4D single-user) or in the 4D Client folder (with 4D Server).

After the file has been saved to the disk, the final name of the file will be returned to the variable passed as the *fileName* parameter. If you attempt to call *IMAP_Download* with a *fileName* that already exists within the download folder, the name will be numerically incremented and its new value as saved to the disk will be returned to the *fileName* variable.

updateSeen is an integer value that indicates if the **¥Seen** flag must be added to the message flags list, whether implicitly or not. This parameter is optional and a default value is used if this parameter is not passed:

- 0 = Add **¥Seen** Flag
- 1 = Do not add **¥Seen** Flag

Default value is set to 0 which implicitly means add **¥Seen** flag.

IMAP_GetCurrentMB

IMAP_GetCurrentMB (imap_ID ; mbName) -> Function result

Parameter	Type		Description
imap_ID	Longint	→	Reference to an IMAP login
mbName	Text	←	Name of the current mailbox
Function result	Integer	↩	Error code

Description

The *IMAP_GetCurrentMB* command returns the current working mailbox name.

imap_ID is a long integer reference to an open connection created with *IMAP_Login*.

mbName returns full name of the current mailbox. If the *mbName* value is a null string, no mailbox is currently selected.

IMAP_GetFlags

IMAP_GetFlags (imap_ID ; startMsg ; endMsg ; msgFlagsArray ; msgNumArray) -> Function result

Parameter	Type		Description
imap_ID	Longint	→	Reference to an IMAP login
startMsg	Longint	→	Start message number
endMsg	Longint	→	End message number
msgFlagsArray	String array	←	Flag values for each message
msgNumArray	Longint array	←	Array of message numbers
Function result	Integer	↩	Error code

Description

The *IMAP_GetFlags* command returns the list of flags for the specified messages.

imap_ID is a long integer reference to an open connection created with *IMAP_Login*.

startMsg is a long integer number that specifies the starting message number of the message range to be examined. The message number is a value representing the position of a message within the list of all messages in the current working mailbox.

endMsg is a long integer number that specifies the ending message number of the message range to be examined. The message number is a value representing the position of a message within the list of all messages in the current working mailbox.

Note: The *IMAP_Delete*, *IMAP_MsgLstInfo*, *IMAP_MsgLst*, *IMAP_SetFlags*, *IMAP_GetFlags* and *IMAP_CopyToMB* commands do not return an error if the *startMsg* is greater than the *endMsg*. In the event that this occurs, the command — in effect — does nothing.

msgFlagsArray is a string or text array returned containing the list of flags, separated by spaces, of each message number between *startMsg* and *endMsg*.

msgNumArray is a long integer array returned containing the message numbers between *startMsg* and *endMsg*.

IMAP_GetMBStatus

IMAP_GetMBStatus (imap_ID ; mbName ; msgNber ; newMsgNber ; unseenMsgNber ; mbUID) -> Function result

Parameter	Type		Description
imap_ID	Longint	→	Reference to an IMAP login
mbName	Text	→	Name of the mailbox
msgNber	Longint	←	Number of messages in the specified mailbox
newMsgNber	Longint	←	Number of messages with the ¥Recent flag set
unseenMsgNber	Longint	←	Number of messages with no ¥Seen flag
mbUID	Longint	←	Specified mailbox unique identifier
Function result	Integer	↩	Error code

Description

The *IMAP_GetMBStatus* command returns the status parameter values of the mailbox specified by *mbName*. It does not change the current mailbox (see *IMAP_SetCurrentMB*), nor does it affect the state of any messages in the specified mailbox (in particular, it usually does not cause messages to lose the **¥Recent** flag, but this can vary depending on the IMAP4 server implementation). This is an alternative used to check mailbox status parameters without deselecting the current mailbox.

This command is particularly useful to:

- Check or retrieve the mailbox unique identifier, and/or,
- Check recent and unseen messages without opening a session for the mailbox.

Important: We strongly recommend that you do not call the *IMAP_GetMBStatus* command using the current mailbox. By doing so, you may encounter problems and the information returned will not necessarily be synchronized with the current mailbox status (in particular for new e-mails).

imap_ID is a long integer reference to an open connection created with *IMAP_Login*.

mbName is the full name of the existing mailbox for which you want to get the status parameter values.

Note: Unlike the *IMAP_ListMBs* command, the *mbName* parameter does not accept wildcards.

msgNber returns the number of messages in the current mailbox (set to zero when the command is called and returns -1 if error).

newMsgNber returns the number of recent messages in the current mailbox (set to zero when the command is called and returns -1 if error).

unseenMsgNber returns the number of unseen messages in the current mailbox (set to zero when the command is called and returns -1 if error)

mbUID returns the mailbox unique identifier validity value (set to zero when the command is called and returns –1 if error).

With the IMAP4 protocol, the mailbox name is not sufficient to identify a mailbox. As such, a unique identifier validity value is associated with each mailbox. This identifier is particularly valuable for synchronizing tasks.
Thus, you can verify if mailbox “A” has been renamed as “B” or deleted, simply by checking the unique identifier validity value.
On the other hand, this identifier allows you to check whether a mailbox named “A” has been deleted and if another “A” mailbox has been created.

IMAP_GetMessage

IMAP_GetMessage (imap_ID ; msgNum ; offset ; length ; msgPart ; msgText ; updateSeen) -> Function result

Parameter	Type		Description
imap_ID	Longint	→	Reference to an IMAP login
msgNum	Longint	→	Message number
offset	Longint	→	Offset of character at which to begin retrieval
length	Longint	→	How many characters to return
msgPart	Integer	→	0 = Entire message, 1 = Only header, 2= Only Body
msgText	Text	←	Message Text
updateSeen	Integer	→	0 = Update ¥Seen Flag; 1 = Do not update
Function result	Integer	↩	Error code

Description

The *IMAP_GetMessage* command returns the complete text of the message identified by *msgNum* within the current mailbox referenced by *IMAP_SetCurrentMB*. Unless otherwise specified by the *IMAP_SetPrefs* command, any linefeed characters within the message will be removed.

The *IMAP_GetMessage* command returns either the entire block of the message, including header information, or with header only or body only, depending on the *msgPart* parameter.

imap_ID is a long integer reference to an open connection created with *IMAP_Login*.

msgNum is a long integer value indicating which message in the mailbox to retrieve. This number represents the position of a message within the current list of messages. You cannot rely on *msgNum* remaining the same for a specific e-mail item from session to session.

offset is a long integer value indicating the number of characters from the beginning of the specified *msgPart* to begin reading. In most circumstances a zero should be passed to this parameter.

length is a long integer value representing the number of characters beyond the *offset* position to retrieve. Since the maximum length of a 4D text variable is limited to 32,000 characters, the *length* parameter should be set to any number below 32,000. Messages whose *msgPart* size is greater than 32K must be saved to the disk via the *IMAP_Download* command.

msgPart indicates the message part to retrieve. Values 0, 1 or 2 can be passed:

- 0 = Entire message,
- 1 = Only header,
- 2 = Only body (means first Text/ plain encountered).

Retrieving entire message or only header retrieves raw text without decoding. On the other hand, when retrieving only body, the text will be decoded and converted automatically if needed (see *POP3_Charset* for more information concerning decoding and conversion rules).

updateSeen is an integer value that indicates if the flag **¥Seen** has to be added to the message flags whether implicitly or not. This parameter is optional and the default value is used if this parameter is not passed.

- 0 = Add **¥Seen** Flag (default value);
- 1= Do not add **¥Seen** Flag;

msgText is a text variable that will receive the retrieved text.

IMAP_GetPrefs

IMAP_GetPrefs (stripLineFeed ; msgFolder) -> Function result

Parameter	Type		Description
stripLineFeed	Integer	⇐	0 = Do not strip LineFeeds, 1 = Strip LineFeeds, -1 = No Change
msgFolder	Text	⇐	Messages folder path
Function result	Integer	↩	Error code

Description

The *IMAP_GetPrefs* command returns the current preferences for the IMAP commands.

The preferences are returned into the variables listed in the parameters.

stripLineFeed returns the current setting of the user's preference for linefeed stripping.

msgFolder is a text variable which returns the local pathname to the default folder in which retrieved messages are stored.

IMAP_ListMBs

IMAP_ListMBs (imap_ID ; mbRefName ; mbName ; mbNamesArray ; mbAttribsArray ; mbHierarArray ; subscribedMB) -> Function result

Parameter	Type		Description
imap_ID	Longint	→	Reference to an IMAP login
mbRefName	Text	→	Null string or Mailbox name or level of mailbox hierarchy
mbName	Text	→	Null string or MailBoxName or wildcards
mbNamesArray	String array	←	Array of mailbox names (pathnames)
mbAttribsArray	String array	←	Array of mailbox attributes
mbHierarArray	String array	←	Array of hierarchy delimiters
subscribedMB	Integer	→	0 = List all available user mailboxes 1 = List only subscribed mailboxes
Function result	Integer	↩	Error code

Description

The *IMAP_ListMBs* command returns the list of available mailboxes for the connected user and attached information. If this command fails, specified arrays are initialized.

mbRefName and *mbName* must be considered together since the resulting Mailbox list will depend on the combination of these two parameter values.

The returned list may be restricted to subscribed mailboxes (see *IMAP_SubscribeMB*) when the last parameter, *subscribedMB*, is set to 1.

When the execution of *IMAP_ListMBs* is very long, either because a large number of mailboxes are being scanned, or because of numerous and complex hierarchical mailbox structures, and so on, you can:

- use wildcards (see below) with *IMAP_ListMBs*,
- or use the *IMAP_ListMBs* command, with the *subscribedMB* parameter set to 1, to list only a set of mailboxes defined using the *IMAP_SubscribeMB* command.

imap_ID is a long integer reference to an open connection created with *IMAP_Login*.

mbRefName is a text value that should be combined with the *mbName* parameter to determine which mailboxes should be looked for. The reference name (*mbRefName*) should be used as a Current Working Directory on Unix systems. In other words, the mailbox name (*mbName*) is interpreted as a file located in the directory specified by the reference name (*mbRefName*). Be sure that the IMAP specification leaves the interpretation of the reference name (*mbRefName*) “implementation-dependent”. We strongly recommend that the user be given an operational mode that does not use any *mbRefName* reference argument. As such, it can interoperate with older servers that did not implement the use of reference arguments.

If *mbRefName* is a null string, only the *mbName* parameter is used to list mailboxes.

If *mbRefName* contains the name of a mailbox or a level of mailbox hierarchy, it should be used to define the context in which the *mbName* parameter must be interpreted.

Note: We strongly recommend that you place a trailing hierarchy delimiter on the reference argument when used. This will ensure full compliance whichever IMAP server is used.

mbName is a text value, intended to be combined with the *mbRefName* parameter, which usually defines the context in which the *mbName* parameter must be interpreted.

If *mbName* is a null string, the hierarchy delimiter is returned.

Note: If you implement a breakout facility using the *mbRefName* parameter, you should allow the user to choose whether or not to use a leading hierarchy delimiter on the mailbox argument. This is because the handling of a leading mailbox hierarchy delimiter varies from one server to another, and even between different electronic mail stores on the same server. In some cases, a leading hierarchy delimiter means “discard the reference argument”; whereas in other cases, the two are concatenated and the extra hierarchy delimiter is discarded.

The *mbNamesArray* array receives the list of available mailboxes’ names.

The *mbAttribsArray* array receives the list of available mailboxes’ attributes.

Mailbox attributes

There are four mailbox attributes defined as follows:

- **¥Noinferiors**: no child levels currently exist and none can be created.
- **¥Noselect**: this name cannot be used as a selectable mailbox.
- **¥Marked**: the server has marked the mailbox as “interesting”; the mailbox probably contains messages added since the last selection.
- **¥Unmarked**: the mailbox does not contain any additional messages since the last selection.

The *mbHierarArray* array receives the list of available mailboxes’ hierarchy delimiters.

The hierarchy delimiter is a character used to delimit hierarchy levels in a mailbox name. You can use it to create child mailboxes and to search higher or lower levels of the naming hierarchy. All children of a top-level hierarchy node use the same separator character.

subscribedMB is an integer value which can be specified when you simply want to list “subscribed” mailboxes. A zero value lists all available user mailboxes. A value of 1 only lists subscribed user mailboxes. *subscribedMB* is an optional parameter that will default to zero if not otherwise specified.

Example 1

The following example:

```
IMAP_ListMBs(imap_ID;"4DIC/Work/";"Test";mbNamesArray;mbAttribsArray;mbHierarArray)
```

...returns all available mailboxes from the “4DIC/Work/Test” mailbox.
Remember that if the IMAP server does not interpret as was intended, do not use the *mbRefName* and concatenate the *mbRefName* and *mbName* values into *mbName*:

```
IMAP_ListMBs(imap_ID;"";"4DIC/Work/Test";mbNamesArray;mbAttribsArray;mbHierarArray)
```

Example 2

The following example:

```
IMAP_ListMBs(imap_ID;"";"";mbNamesArray;mbAttribsArray;mbHierarArray)
```

...returns the hierarchy delimiter.

Using the Wildcard character

You can use wildcards in the *mbRefName* and *mbName* parameters to make mailbox selection easier. You will find an example of current wildcards below, but please note that the interpretation of wildcards will depend on the IMAP server; consequently, these examples may not work. In this case, check your IMAP server wildcards.

- “ * ” matches zero or more characters in its position:

```
IMAP_ListMBs(imap_ID;"";"*";mbNamesArray;mbAttribsArray;mbHierarArray)
```

... returns all mailboxes available to the connected user.

```
IMAP_ListMBs(imap_ID;"";"Work*";mbNamesArray;mbAttribsArray;mbHierarArray)
```

... returns all available mailboxes matching the root “Work”.

- “ % ” is similar to “ * ”, but it does not match a hierarchy delimiter. If the “%” wildcard is the last character of the *mbName* parameter, matching hierarchy levels are also returned. If these hierarchy levels are not selectable mailboxes, they are returned with the **¥Noselect** mailbox attribute (see paragraph “Mailbox attributes”).

```
IMAP_ListMBs(imap_ID"";"Work/%";mbNamesArray;mbAttribsArray;mbHierarArray)
```

... returns all mailboxes matching the root “Work”, plus one hierarchy level available for the connected user.

“%” can be helpful in order to parse the mailbox hierarchy level by level.
Given the following mailbox hierarchy:

```
INBOX
   MailboxA
      MailboxAA
      MailboxAB
   MailboxB
      MailboxBA
      MailboxBB
   MailboxC
      MailboxCA
      MailboxCB
```

```
IMAP_ListMBs(imap_ID;"";"%";mbNamesArray;mbAttribsArray;mbHierarArray)
```

... returns INBOX, MailboxA, MailboxB and MailboxC.

```
IMAP_ListMBs(imap_ID;"";"MailboxA%";mbNamesArray;mbAttribsArray;mbHierarArray)
```

... returns MailboxAA and MailboxAB.

Using this technique, you can give the user complete flexibility without being bogged down by the voluminous reply to

```
IMAP_ListMBs(imap_ID;"";"*";mbNamesArray;mbAttribsArray;mbHierarArray)
```

Note that IMAP servers themselves may limit the number of levels to be scanned.

IMAP_Login

IMAP_Login (hostName ; userName ; password ; imap_ID {; sessionParam}) -> Function result

Parameter	Type		Description
hostName	String	→	Host Name or IP address of the IMAP server
userName	String	→	User name
password	String	→	Password
imap_ID	Longint	←	Reference to this IMAP login
sessionParam	Longint	→	1 = Use SSL, 0 or omitted = Do not use SSL
Function result	Integer	↩	Error code

Description

The *IMAP_Login* command logs the user onto the IMAP electronic mail server with the given user name and password.

This particular login is given a connection reference (*imap_ID*) to which subsequent IMAP commands can refer.

The connection is closed using the *IMAP_Logout* command or when the IMAP server inactivity timer has timed out.

hostName is the host name or IP address of the IMAP electronic mail server. It is recommended that the host name be used but, if needed, an IP address may be used.

userName is the user's name on the IMAP electronic mail server. The *userName* should not contain the domain. For example, for the address “jack@4d.com”, the *userName* would be just “jack”.

password is the password for the *userName* on the IMAP electronic mail server.

imap_ID is a long integer variable into which a reference to the connection just established is returned. This parameter must be passed a 4D variable in order to accept the returned results. The variable will be used in all subsequent commands which perform actions related to this session. If *IMAP_Login* fails, *imap_ID* is set to zero.

The optional *sessionParam* parameter enables the SSL protocol for a connection:

- If you pass 1, the connection to the IMAP server will be made in SSL (synchronous mode),
- If you pass 0 or omit this parameter, the connection will be made in standard, non-secure mode.

Example

Here is a typical connection sequence:

```
$ErrorNum:=IMAP_Login(vHost;vUserName;vUserPassword;vImap_ID;1)
If($ErrorNum =0)
   C_TEXT(vCapability)
   $ErrorNum:=IMAP_Capability(vImap_ID;vCapability))
 ` IMAP commands using vImap_ID parameter
End if
$ErrorNum:=IMAP_Logout(vImap_ID)
```

IMAP_Logout

IMAP_Logout (imap_ID) -> Function result

Parameter	Type		Description
imap_ID	Longint	→	Reference to an IMAP login
		←	0 = Command successfully logs off
Function result	Integer	↩	Error code

Description

The *IMAP_Logout* command will log off of the open IMAP connection referred to by the *imap_ID* variable. If the command successfully logs off the IMAP server, a zero value is returned back as the current *imap_ID*.

Note: Closing a connection automatically closes the current session.

imap_ID is a long integer reference to an open connection created with *IMAP_Login*.

IMAP_MsgFetch

IMAP_MsgFetch (imap_ID ; msgNum ; msgDataItem ; msgDataItemValue) -> Function result

Parameter	Type		Description
imap_ID	Longint	⇒	Reference to an IMAP login
msgNum	Longint	⇒	Message number
msgDataItem	Text	⇒	Data item(s) to retrieve
msgDataItemValue	Text	⇐	Data item(s) value
Function result	Integer	↩	Error code

Description

The *IMAP_MsgFetch* command allows the user to request one or several basic data items for the specified message without downloading the message.

imap_ID is a long integer reference to an open connection created with *IMAP_Login*.

msgNum is a long integer value indicating the message to be checked. This value represents the position of a message within the current list of messages. You cannot rely on the *msgNum* remaining the same for a specific e-mail item from session to session.

msgDataItem is a text variable that indicates one or several data items you want to retrieve. In the case of several data items, a space character must separate each of them. There are two kinds of data items:

- Basic data items, which only retrieve one piece of information, and
- Macro data items, which retrieve several pieces of information, issued from basic data items at one time. Three macros specify commonly-used sets of data items and can be used in place of them. A macro must be used by itself, and not in conjunction with other macros or data items.

For more information on data items, see the “Basic data items” and “Macro data items” paragraphs below.

msgDataItemValue is a text variable that can return either a single *DataItem/DataItemValue* pair or a list of *DataItem/DataItemValue* pairs depending on the *msgDataItem* parameter value.

- In the case of a single *DataItem/DataItemValue*, the returned text structure is as follows: *DataItem name+Space+DataItemValue*
- In the case of a list of *DataItem/DataItemValue* pairs, the returned text structure is as follows: *DataItem name1+Space+DataItemValue1+Space+DataItem name2+Space+DataItemValue2*.

msgDataItemValue may contain a parenthesized list, a quoted string or a single string depending on the *msgDataItem* parameter.

- The parenthesized list is structured as follows (see **FLAGS** case for example): (*FirstDataItemValue+space+2ndDataItemValue*)
  If the parenthesized list returns only parentheses, this means that there is no item value. This rule does not apply to address parenthesized lists (see **ENVELOPE**).
- Quoted strings are structured as follows (see **INTERNALDATE** case for example): *DataItem name+Space+Quote+DataItemValue+Quote*
  If the DataItem value returns "", this means that it is a null string.
- Strings that are not quoted indicate integer, long integer or numeric values and are structured as follows: *DataItem name+Space+DataItemValue*.
  In this case, you will most likely have to convert to the appropriate type (see **UID** case for example).

Note: Quotes are generally used when the string value includes special characters, such as a space or parentheses. As such, when you parse the resulting string of the **IMAP_Fetch** command, quote characters are taken into consideration when processing the string content.

Basic data items

- **INTERNALDATE**
  Retrieves the internal date and time of the message on the IMAP server. This is not the date and time returned by the “Date” header, but rather a date and time that indicate when the message was received. For messages delivered via an SMTP server, this date usually reflects the date and time of the final delivery of the message. For messages sent after an **IMAP_Copy** command, this data usually reflects the internal date and time of the source message.
  INTERNALDATE data item value returns a quoted string.

Example:

```
msgDataItem:="INTERNALDATE"
$Err:=IMAP_MsgFetch(imap_ID;1;msgDataItem;msgDataItemValue)
```

msgDataItem returns INTERNALDATE "17-Jul-2001 15:45:37 +0200"

- **FLAGS**
  Retrieves the parenthesized list of flags that are set for the specified message. Flags values are separated by spaces.

Example:

```
msgDataItem:="FLAGS"
$Err:=IMAP_MsgFetch(imap_ID;1;msgDataItem;msgDataItemValue)
```

msgDataItem returns FLAGS () if there is no flag set for the specified message.
msgDataItem returns FLAGS (¥Seen ¥Answered) if **¥Seen** and **¥Answered** flags are set for the specified message.

- **RFC822.SIZE**
  Retrieves the number of bytes in the message, as expressed in RFC-822 format. The Data item is separated from the returned value by a space. An unquoted string is returned, which means you will probably need to convert this string into a longint value (see **UID** example).

Example:

```
msgDataItem:="RFC822.SIZE"
$Err:=IMAP_MsgFetch(imap_ID;1;msgDataItem;msgDataItemValue)
```

msgDataItem returns RFC822.SIZE 99599

- **ENVELOPE**
  Retrieves the parenthesized list describing the header part for the specified message. The server computes this by parsing the message header and defaulting various fields where necessary.
  The header fields are returned in the following order: date, subject, from, sender, reply-to, to, cc, bcc, in-reply-to, and message-id. The date, subject, in-reply-to and message-id fields are quoted strings: ENVELOPE ("date" "subject" (from) (sender) (reply-to) (to) (cc) (bcc) "in-reply-to" "message-id")
  Example:

```
msgDataItem:="ENVELOPE"
$Err:=IMAP_MsgFetch(imap_ID;1;msgDataItem;msgDataItemValue)
```

msgDataItem returns ENVELOPE ("Tue, 17 Jul 2001 17:26:34 +0200" "Test" (("RSmith" NIL "RSmith" "test")) (("RSmith" NIL "RSmith" "test")) (("RSmith" NIL "RSmith" "test")) (("RSmith" NIL "RSmith" "test")) () () "" "<ee6b33a.-1@Mail.x6foadRIbnm>")

Date:	"Tue, 17 Jul 2001 17:26:34 +0200"	date header
Subject:	"Test"	subject header
From:	(("RSmith" NIL "RSmith" "test"))	address structures
Sender:	(("RSmith" NIL "RSmith" "test"))	address structures
reply-to:	(("RSmith" NIL "RSmith" "test"))	address structures
to:	(("RSmith" NIL "RSmith" "test"))	address structures
cc:	()	Cc header not used
bcc:	()	Bcc header not used
in-reply-to:	""	In-reply-to header
message-id:	"<ee6b33a.-1@Mail.x6foadRIbnm>"	message-id header

The from, sender, reply-to, to, cc and bcc fields are parenthesized lists of address structures. An address structure is a parenthesized list that describes an electronic mail address. The fields of an address structure are in the following order: personal name, [SMTP] at-domain-list (source route), mailbox name and host name. For instance, *(("RSmith" NIL "RSmith" "test"))*.

(From [RFC-822]) Group syntax is indicated by a special form of address structure where the host name field is NIL. If the mailbox name field is also NIL, this is an end-of-group marker (semi-colon in RFC 822 syntax). If the mailbox name field is non-NIL, this is the start-of-group marker and the mailbox name field holds the group name phrase.
A field of an envelope or address structure that is not applicable is presented as NIL. Note that the server MUST default the reply-to and sender fields from the “from” field; the client is not expected to know how to do this.

- **BODY**
  Returns the same information as **BODYSTRUCTURE** except for the Extension data (see **BODYSTRUCTURE**) which is not returned.
  Example:

```
msgDataItem:="BODY"
$Err:=IMAP_MsgFetch(imap_ID;1;msgDataItem;msgDataItemValue)
```

msgDataItem returns BODY ("TEXT" "PLAIN" ("CHARSET" "us-ascii") NIL NIL "8BIT" 8 1)

- **BODYSTRUCTURE**
  Retrieves the MIME body structure of the message. The server computes this by parsing the MIME header fields in the message header and MIME headers in the body part. This data item is especially useful for scanning a message content without downloading it. For instance, you can quickly check the size of each part or just check the attachment file names. **BODYSTRUCTURE** returns a parenthesized list including the parenthesized list, quoted strings and unquoted strings.
  Depending on the message content, **BODYSTRUCTURE** will return either a “non- multipart” parenthesized list or a nested one (“multipart” parenthesized list):
  - “**non-multipart**” parenthesized list: this is, for instance, similar to non-multipart e-mail; a simple text message of 48 lines and 2279 bytes can have a body structure of: *("TEXT" "PLAIN" ("CHARSET" "us-ascii") NIL NIL "8BIT" 8 1 NIL NIL NIL).*
    The basic fields of a “non-multipart” parenthesized list are in the following order:

body type	A string giving the content media type name (Content-type: media type e.g. TEXT)
body subtype	A string giving the content subtype name (Content-type: subtype e.g. PLAIN)
body parameter	A parenthesized list of attribute/value pairs
parenthesized list	[e.g. ("CHARSET" "US-ASCII" "NAME" "cc.diff") where "US-ASCII" is the value of "CHARSET" and "cc.diff" is the value of "NAME".
body id	A string giving the content id (allows one body to make a reference to another one). Accordingly, bodies may be labeled using the "Content-ID" header field. The Content-ID value has special semantics in the case of a multipart/alternative media type. This is explained in the section of RFC 2046 dealing with multipart/alternative cases.
body description	A string giving the content description
body encoding	A string giving the content transfer encoding (Content-Transfer-Encoding)
body size	A number giving the size of the body in bytes. Note that this is the size during transfer encoding and not the resulting size after decoding.

- A body of the MESSAGE type and RFC822 subtype contains, immediately following the basic fields, the envelope structure, body structure, and size in text lines of the encapsulated message.
- A body of the TEXT type contains, immediately following the basic fields, the size of the body in text lines. Note that this is the size during content transfer encoding and not the resulting size after decoding.
Extension data follows the basic fields and type-specific fields listed above. Extension data is never returned with the **BODY** fetch, but can be returned with a **BODYSTRUCTURE** fetch.
Extension data , if present, of a “non multipart” parenthesized list MUST be in the defined order:

body MD5	A string giving the body MD5 value as defined in [MD5]
body disposition	A parenthesized list consisting of a disposition type string followed by a parenthesized list of disposition attribute/value pairs as defined in [DISPOSITION]
body language	A string or parenthesized list giving the body language value as defined in [LANGUAGE-TAGS]

Any extension data which follows are not yet defined in this version of the protocol and will be as described above under multipart extension data.
Example: *("TEXT" "PLAIN" ("CHARSET" "US-ASCII") NIL NIL "7BIT" 2279 48 NIL NIL NIL)*
Description:*("bodytype" "bodysubtype" (BodyParameterParenthesizedList) bodyId bodyDescription "bodyEncoding" BodySize BodySizeInTextLines ExtensionDataBODYmd5 ExtensionDataBodyDisposition ExtensionDataBodyLanguage)*

- “**multipart**” parenthesized list: this is the case of multipart e-mail; it includes a “non-multipart” parenthesized list.
Parenthesis nesting indicates multiple parts. The first element of the parenthesized list is a nested body instead of a body type. The second element of the parenthesized list is the multipart subtype (mixed, digest, parallel, alternative, etc.).
The multipart subtype is followed by the Extension data. Extension data, if present, MUST be in the defined order:

body parameter	A parenthesized list of attribute/value pairs parenthesized list
body disposition	A parenthesized list consisting of a disposition type string followed by a parenthesized list of disposition attribute/value pairsas defined in [DISPOSITION]
body language	A string or parenthesized list giving the body language value as defined in [LANGUAGE-TAGS]

Any extension data which follows are not yet defined in this version of the protocol. This extension data can consist of zero or more NILs, strings, numbers or potentially nested parenthesized lists of such data. Client implementations that do a **BODYSTRUCTURE** fetch MUST be prepared to accept such extension data. Server implementations MUST NOT send such extension data until it has been defined by a revision of this protocol.

Example: *BODYSTRUCTURE (("TEXT" "PLAIN" ("CHARSET" "us-ascii") NIL NIL "7BIT" 22 1 NIL NIL NIL)("APPLICATION" "BYTE-STREAM" ("NAME" "casta37.jpg" "X-MAC-TYPE" "4A504547" "X-MAC-CREATOR" "6F676C65") NIL NIL "BASE64" 98642 NIL ("ATTACHMENT" ("FILENAME" "casta37.jpg")) NIL) "MIXED" ("BOUNDARY" "4D_====================1385356==") NIL NIL)*
Description: *(("bodytype" "bodysubtype" (BodyParameterParenthesizedList) bodyId bodyDescription "bodyEncoding" BodySize BodySizeInTextLines ExtensionDataBODYmd5 ExtensionDataBodyDisposition ExtensionDataBodyLanguage) ("bodytype" "bodysubtype" (BodyParameterParenthesizedList) bodyId bodyDescription "bodyEncoding" BodySize BodySizeInTextLines ExtensionDataBODYmd5 ExtensionDataBodyDisposition ExtensionDataBodyLanguage) "multipartSubtype" (ExtensionDataBodyParameterList) ExtensionDataBodyDisposition ExtensionDataBodyLanguage))*

- **UID**
  Retrieves a number expressing the message unique identifier of the message. This is equivalent to executing the *IMAP_UIDToMsgNum*. Since this number is returned into a text area, you will have to convert it into a Long integer.
  Example:

```
msgDataItem:="UID"
$Err:=IMAP_MsgFetch(imap_ID;1;msgDataItem;msgDataItemValue)
```

msgDataItemValue returns UID 250000186

To retrieve a longint value:

```
C_LONGINT(vLongint)
VLongint:=Num("250000186")
```

Macro data items

- **FAST**
  Macro equivalent to: (FLAGS INTERNALDATE RFC822.SIZE)
  Example:

```
$Err:=IMAP_MsgFetch(imap_ID;msgNum;"FAST";msgDataItemValue)
```

msgDataItemValue returns *"FLAGS (¥Seen ¥Answered) INTERNALDATE "17-Jul-2001 15:45:37 +0200" RFC822.SIZE 99599"*

- **ALL**
  Macro equivalent to: *(FLAGS INTERNALDATE RFC822.SIZE ENVELOPE)*
- **FULL**
  Macro equivalent to: *(FLAGS INTERNALDATE RFC822.SIZE ENVELOPE BODY)*

IMAP_MsgInfo

IMAP_MsgInfo (imap_ID ; msgNum ; msgSize ; uniqueID) -> Function result

Parameter	Type		Description
imap_ID	Longint	→	Reference to an IMAP login
msgNum	Longint	→	Message number
msgSize	Longint	←	Message size
uniqueID	Longint	←	Unique ID of message on server
Function result	Integer	↩	Error code

Description

The *IMAP_MsgInfo* command returns information about the message identified by *msgNum* within the currently selected mailbox. Information about the size of the message and its Unique ID will be returned.

imap_ID is a long integer reference to an open connection created with *IMAP_Login*.

msgNum is a long integer value indicating which message in the mailbox you wish to retrieve information about. The *msgNum* represents the position of a message within the current list of messages. You cannot rely on the *msgNum* to remain the same for a specific e-mail item from session to session.

msgSize is the long integer value returned containing the message size referenced by *msgNum*.

uniqueID is a long integer variable denoting the Unique ID of the message on the server. The *uniqueID* is a value assigned to the message by the IMAP4 server software. This value will not change from session to session in the same way as msgNum. The *uniqueID* value is a good reference to verify if your database has already downloaded a message from the server

IMAP_MsgLst

IMAP_MsgLst (imap_ID ; startMsg ; endMsg ; msgHeaderArray ; msgNumArray ; msgIdArray ; msgValueArray) -> Function result

Parameter	Type		Description
imap_ID	Longint	⇒	Reference to an IMAP login
startMsg	Longint	⇒	Start message number
endMsg	Longint	⇒	End message number
msgHeaderArray	String array	⇒	Array of headers to retrieve
msgNumArray	Longint array	⇐	Array of message numbers
msgIdArray	Longint array	⇐	Array of Unique Msg IDs
msgValueArray	2D String array, 2D Text array	⇐	2D Array of header values
Function result	Integer	↩	Error code

Description

The *IMAP_MsgLst* command is used to get specific information of mailbox contents. It allows the user to request specific columns of the message list. This command can only return header item values; it cannot be used to retrieve the body of a message. Header content is automatically decoded and converted if needed (see *POP3_Charset* for more information concerning the decoding and conversions rules).

imap_ID is a long integer reference to an open connection created with *IMAP_Login*.

startMsg is a long integer number that specifies the starting message number of the message range to be examined. The message number is a value representing the position of a message within the list of all messages in the current working mailbox.

endMsg is a long integer number that specifies the ending message number of the message range to be examined. The message number is a value representing the position of a message within the list of all messages in the current working mailbox.

Note: The *IMAP_Delete*, *IMAP_MsgLstInfo*, *IMAP_MsgLst*, *IMAP_SetFlags*, *IMAP_GetFlags* and *IMAP_CopyToMB* commands do not return an error if the *startMsg* is greater than the *endMsg*. In the event that this occurs, the command – in effect – does nothing.

msgHeaderArray is a string or text array that lists the specific e-mail headers you wish to retrieve.

msgNumArray is a long integer array returned containing the message numbers between *startMsg* and *endMsg*.

msgIdArray is a long integer array returning the Unique IDs of the messages between *startMsg* and *endMsg*.

msgValueArray is a 2-dimensional array that receives the data for each header specified in *msgHeaderArray*. Each requested header will have a matching array in the first dimension of *valueArray*.

Example

```
aHeaders{1}:="Date:"
aHeaders{2}:="From:"
aHeaders{3}:="Subject:"
IMAP_MsgLst(IMAP_ID;vStart;vEnd;aHeaders;aMsgNum;aMsgId;aValues)
```

aValues{1}{1} may equal “Thu, 19 November 1998 00:24:02 -0800”
aValues{2}{1} may equal “Jack@4d.com”
aValues{3}{1} may equal “Call your wife”

Errors are handled the following manner:

1) Only communication-related error codes will be returned. If the command cannot complete its task because of an error (network, syntax, server, etc.) then the appropriate error code will be returned.

2) If a message within the specified range of messages does not exist or gets an error:

- No array element is created for that message.

- No Error code will be returned.

3) The inability to locate any or all of the specified headers within a message does not constitute an error:

- An array element for the message will be created.
- The *aMsgNum* and *aMsgID* array elements will contain the appropriate values.
- For each header which does not exist in the message, a null string will be returned into that array element.
- No Error code will be returned.

IMAP_MsgLstInfo

IMAP_MsgLstInfo (imap_ID ; startMsg ; endMsg ; msgSizeArray ; msgNumArray ; msgIdArray) -> Function result

Parameter	Type		Description
imap_ID	Longint	⇒	Reference to an IMAP login
startMsg	Longint	⇒	Start message number
endMsg	Longint	⇒	End message number
msgSizeArray	Longint array	⇐	Array of sizes
msgNumArray	Longint array	⇐	Array of message numbers
msgIdArray	Longint array	⇐	Array of Unique Msg IDs
Function result	Integer	↩	Error code

Description

The *IMAP_MsgLstInfo* command returns information about a set of messages in the current working mailbox (defined by the *IMAP_SetCurrentMB* command). The information is returned into three arrays with each element of the arrays corresponding to one message. Information is returned about the message size and number. The arrays passed as parameters must be of pre-declared types, though they may be of any size. The *IMAP_MsgLstInfo* command will reset the size of each array to the number of messages retrieved.

The *IMAP_MsgLstInfo* command will not return an error number if it fails to retrieve information on a message within the current message list. If an error occurs, no element is created in the arrays for the problem message. If the command reads each message successfully, the *msgNumArray* should contain numeric values in a sequential order. If problems were encountered, there may be gaps in the sequence of numbers held in *msgNumArray*.

imap_ID is a long integer reference to an open connection created with *IMAP_Login*.

startMsg is a long integer number that specifies the starting message number of the message range to be examined. The message number is a value representing the position of a message within the list of all messages in the current working mailbox.

endMsg is a long integer number which specifies the ending message number of the message range to be examined. The message number is a value representing the position of a message within the list of all messages in the current working mailbox.

Note: The *IMAP_Delete*, *IMAP_MsgLstInfo*, *IMAP_MsgLst*, *IMAP_SetFlags*, *IMAP_GetFlags* and *IMAP_CopyToMB* commands do not return an error if the *startMsg* is greater than the *endMsg*. In the event that this occurs, the command – in effect – does nothing.

sizeArray is a long integer array returned containing the sizes of each message between *startMsg* and *endMsg*.

msgNumArray is a long integer array returned containing the message numbers between *startMsg* and *endMsg*.

msgIdArray is a long integer array returning the Unique IDs of the messages between *startMsg* and *endMsg*.

IMAP_MsgNumToUID

IMAP_MsgNumToUID (imap_ID ; msgNum ; unique_ID) -> Function result

Parameter	Type		Description
imap_ID	Longint	→	Reference to an IMAP login
msgNum	Longint	→	Message number
unique_ID	Longint	←	Message unique identifier value
Function result	Integer	↩	Error code

Description

The *IMAP_MsgNumToUID* command converts a message number within the list of messages in the current mailbox referenced by *imap_ID* to its current *unique_ID* value.

imap_ID is a long integer reference to an open connection created with *IMAP_Login*.

msgNum is a long integer returned containing the current message number (its position within the current message list) of the item identified by *unique_ID*. If the *unique_ID* cannot be found on the server, a zero is returned in *msgNum* and no error is returned.

unique_ID is a long integer value returning the Unique ID of a message to be located on the IMAP server.

IMAP_RenameMB

IMAP_RenameMB (imap_ID ; mbName ; newMBName) -> Function result

Parameter	Type		Description
imap_ID	Longint	⇨	Reference to an IMAP login
mbName	Text	⇨	Name of the mailbox to be renamed
newMBName	Text	⇨	New mailbox name
Function result	Integer	⮌	Error code

Description

The *IMAP_RenameMB* command changes the name of a mailbox. Attempting to rename a mailbox from a mailbox name that does not exist or to a mailbox name that already exists will generate an error.

Note: Renaming an INBOX is permitted and involves special behavior. It moves all messages in the INBOX to a new mailbox with the given name, leaving the INBOX empty. If the server implementation allows child names for the INBOX, these are unaffected by its renaming.

imap_ID is a long integer reference to an open connection created with *IMAP_Login*.

mbName is the full name of the mailbox to be renamed (see naming rules in the IMAP introduction).

newMBName is the full name to apply to rename the *mbName* mailbox.

IMAP_Search

IMAP_Search (imap_ID ; searchCriteria ; msgNumArray) -> Function result

Parameter	Type		Description
imap_ID	Longint	⇒	Reference to an IMAP login
searchCriteria	Text	⇒	Search criteria
msgNumArray	Longint array	⇐	Array of message numbers
Function result	Integer	↩	Error code

Description

The *IMAP_Search* command searches for messages that match the given search criteria in the current mailbox. *searchCriteria* consists of one or more search keys. *msgNumArray* returns a listing of message sequence numbers corresponding to those messages that match the search criteria.
msgNumArray returns a listing of message sequence numbers corresponding to those messages that match the searching criteria.

imap_ID is a long integer reference to an open connection created with *IMAP_Login*.

searchCriteria is a text parameter listing one or more search keys (see "Authorized search keys" at the end of this paragraph) associated or not with values to look for. A search key may be a single item or can also be a parenthesized list of one or more search keys. For example:

SearchKey1 = FLAGGED
SearchKey2 = NOT FLAGGED
SearchKey3 = FLAGGED DRAFT

Note: Matching is usually not case-sensitive.

- If the *searchCriteria* is a null string, the search will be equivalent to a "select all":

  ```
  IMAP_Search(imap_ID;"";msgNumArray)
  ```

  ... returns all messages in the current working mailbox.

- If the *searchCriteria* includes multiple search keys, the result is the intersection (AND function) of all the messages that match those keys.
  searchCriteria = FLAGGED FROM "SMITH"
  ... returns all messages with **¥Flagged** flag set AND sent by Smith.

- You can use the **OR** or **NOT** operators as follows:
  searchCriteria = OR SEEN FLAGGED
  ... returns all messages with **¥Seen** flag set OR **¥Flagged** flag set
  searchCriteria = NOT SEEN
  ... returns all messages with **¥Seen** flag not set .
  searchCriteria = HEADER CONTENT-TYPE "MIXED" NOT HEADER CONTENT-TYPE "TEXT"...
  ... returns message whose content-type header contains "Mixed" and does not contain "Text".
  searchCriteria = HEADER CONTENT-TYPE "E" NOT SUBJECT "o" NOT HEADER CONTENT-TYPE "MIXED"
  ... returns message whose content-type header contains " e " and whose Subject header does not contain " o " and whose content-type header is not " Mixed ".
  searchCriteria = OR (ANSWERED SMALLER 400) (HEADER CONTENT-TYPE "E" NOT SUBJECT "o" NOT HEADER CONTENT-TYPE "MIXED")
  ... returns messages matching the first parenthesized list OR the second one.
  searchCriteria = OR ANSWERED SMALLER 400 (HEADER CONTENT-TYPE "E" NOT SUBJECT "o" NOT HEADER CONTENT-TYPE "MIXED")
  ... returns messages matching with **¥Answered** flag set OR messages whose size is smaller than 400 bytes AND the parenthesized list as specified.

As concerns the last two examples, notice that the result of the search is different when you remove the parentheses of the first search key list.

- The *searchCriteria* may include the optional *[CHARSET]* specification. This consists of the "CHARSET" word followed by a registered *[CHARSET] (US ASCII, ISO-8859)*. It indicates the charset of the *searchCriteria* string. Therefore, you must convert the *searchCriteria* string into the specified charset if you use the *[CHARSET]* specification (see the 4D **Mac to ISO** command).
  By default, 4D Internet Commands encode in Quotable Printable the *searchCriteria* string if it contains extended characters.
  searchCriteria = CHARSET "ISO-8859" BODY "Help"
  ... means the search criteria uses the charset iso-8859 and the server will have to convert the search criteria before searching, if necessary.

Search value types

Search-keys may request the value to search for:

- **Search-keys with a date value**

<date> is a string that must be formatted as follows: date-day+"-"+date-month+"-"+date-year where date-day indicates the number of the day of the month (max. 2 characters), date-month indicates the name of the month (Jan/Feb/Mar/Apr/May/Jun/Jul/Aug/Sep/Oct/Dec) and date-year indicates the year (4 characters).

Example: *searchCriteria = SENTBEFORE 1-Feb-2000* (a date does not usually need to be quoted since it does not contain any special characters)

- **Search-keys with a string value**

<string> may contain any character and must be quoted. If the string does not contain any special characters, like the space character for instance, it does not need to be quoted. Quoting such strings will ensure that your string value will be correctly interpreted.

Example: *searchCriteria = FROM "SMITH"*

Note: For all search keys that use strings, a message matches the key if the string is a substring of the field. Matching is not case-sensitive.

- **Search-keys with a field-name value**

<field-name> is the name of a header field.

Example: *searchCriteria = HEADER CONTENT-TYPE "MIXED"*

- **Search-keys with a flag value**

<flag> may accept one or several keywords (including standard flags), separated by spaces.

Example: *searchCriteria = KEYWORD ¥Flagged ¥Draft*

- **Search-keys with a message set value**

Identifies a set of messages. For message sequence numbers, these are consecutive numbers from 1 to the total number of messages in the mailbox.
A comma delimits individual numbers; a colon delimits between two numbers inclusive.

Examples:
2,4:7,9,12:* is 2,4,5,6,7,9,12,13,14,15 for a mailbox with 15 messages.

searchCriteria = 1:5 ANSWERED search in message selection from message sequence number 1 to 5 for messages which have the **¥Answered** flag set.

searchCriteria= 2,4 ANSWERED search in the message selection (message numbers 2 and 4) for messages which have the **¥Answered** flag set.

Authorized search-keys

ALL: All messages in the mailbox.

ANSWERED: Messages with the **¥Answered** flag set.

UNANSWERED: Messages that do not have the **¥Answered** flag set.

DELETED: Messages with the **¥Deleted** flag set.

UNDELETED: Messages that do not have the **¥Deleted** flag set.

DRAFT: Messages with the **¥Draft** flag set.

UNDRAFT: Messages that do not have the **¥Draft** flag set.

FLAGGED: Messages with the **¥Flagged** flag set.

UNFLAGGED: Messages that do not have the **¥Flagged** flag set.

RECENT: Messages that have the **¥Recent** flag set.

OLD: Messages that do not have the **¥Recent** flag set.

SEEN: Messages that have the **¥Seen** flag set.

UNSEEN: Messages that do not have the **¥Seen** flag set.

NEW: Messages that have the **¥Recent** flag set but not the **¥Seen** flag. This is functionally equivalent to "(RECENT UNSEEN)".

KEYWORD <flag>: Messages with the specified keyword set.

UNKEYWORD <flag>: Messages that do not have the specified keyword set.

BEFORE <date>: Messages whose internal date is earlier than the specified date.

ON <date>: Messages whose internal date is within the specified date.

SINCE <date>: Messages whose internal date is within or later than the specified date.

SENTBEFORE <date>: Messages whose Date header is earlier than the specified date.

SENTON <date>: Messages whose Date header is within the specified date.

SENTSINCE <date>: Messages whose Date header is within or later than the specified date.

TO <string>: Messages that contain the specified string in the TO header.

FROM <string>: Messages that contain the specified string in the FROM header.

CC <string>: Messages that contain the specified string in the CC header.

BCC <string>: Messages that contain the specified string in the BCC header.

SUBJECT <string>: Messages that contain the specified string in the Subject header.

BODY <string>: Messages that contain the specified string in the message body.

TEXT <string>: Messages that contain the specified string in the header or in the message body.

HEADER <field-name> <string>: Messages that have a header with the specified field-name and that contain the specified string in the field-body.

UID <message UID>: Messages with unique identifiers corresponding to the specified unique identifier set.

LARGER <n>: Messages with a size larger than the specified number of bytes.

SMALLER <n>: Messages with a size smaller than the specified number of bytes.

NOT <search-key>: Messages that do not match the specified search key.

OR <search-key1> <search-key2>: Messages that match either search key.

IMAP_SetCurrentMB

IMAP_SetCurrentMB (imap_ID ; mbName ; msgNber ; newMsgNber ; customFlags ; permanentFlags ; mbUID) -> Function result

Parameter	Type		Description
imap_ID	Longint	→	Reference to an IMAP login
mbName	Text	→	Name of the mailbox to select
msgNber	Longint	←	Number of messages in the current mailbox
newMsgNber	Longint	←	Number of messages with the ¥Recent flag set
customFlags	Text	←	List of flags currently used for the mailbox
permanentFlags	Text	←	List of flags permanently modifiable
mbUID	Longint	←	Mailbox unique identifier value
Function result	Integer	↩	Error code

Description

The *IMAP_SetCurrentMB* command allows you to open a session (i.e. selects the current working Mailbox) in order to manage the messages of the specified mailbox.

Only one session can be opened at a time during a connection; simultaneous access to multiple mailboxes requires multiple connections (multiple *IMAP_Login*). The *IMAP_SetCurrentMB* command automatically closes the current session before attempting the new selection. Consequently, if a mailbox is defined as current and an *IMAP_SetCurrentMB* command fails, there will no longer be any mailbox defined as current.

You can close a session, (i.e. close the current mailbox) without selecting a new one, by executing the *IMAP_SetCurrentMB* command using a non-existing mbName and while managing the returned error, either by executing the *IMAP_CloseCurrentMB*, or by executing the *IMAP_Logout* command.

imap_ID is a long integer reference to an open connection created with *IMAP_Login*.

mbName is the full name of an existing mailbox to be defined as current.

msgNber returns the number of messages in the current mailbox (set to zero when *IMAP_SetCurrentMB* is called and returns -1 if error).

newMsgNber returns the number of recent messages in the current mailbox (set to zero when *IMAP_SetCurrentMB* is called and returns -1 if error).

customFlags returns the complete list of flags used in the current mailbox. Note that only flags listed in the *permanentFlags* string can be modified.

permanentFlags returns the list of mailbox message flags that can be changed permanently (except for the **¥Recent** flag, which is managed by the IMAP server). (Set to null string when *IMAP_SetCurrentMB* is called). Note that the *permanentFlags* string can also include the special flag ¥*, which means that keywords can be created by trying to store those flags in the mailbox (see *IMAP_SetFlags*).

If *permanentFlags* returns a null string, this means that all the flags listed in the *customFlags* parameter can be permanently changed.

mbUID returns a unique identifier validity value for the current mailbox.
This identifier can be particularly useful if a mailbox is deleted and a new mailbox with the same name is created at a later date. Since the name is the same, a client may not know that this is a new mailbox unless the unique identifier validity is different.

IMAP_SetFlags

IMAP_SetFlags (imap_ID ; startMsg ; endMsg ; msgFlagsList ; deleteOption) -> Function result

Parameter	Type		Description
imap_ID	Longint	→	Reference to an IMAP login
startMsg	Longint	→	Start message number
endMsg	Longint	→	End message number
msgFlagsList	String	→	Flag values to add or remove
deleteOption	Integer	→	1 = add flag value, 0 = remove flag value
Function result	Integer	←	Error code

Description

The *IMAP_SetFlags* command allows adding or clearing of several flags at a time, attached to the specified range of messages.

IMAP protocol allows associating of a list of flags with a message. There are two types of flags: **permanent** or **session-only**.
Permanent flags are added or removed permanently from the message flags (see *IMAP_SetCurrentMB*); in other words, subsequent sessions will reflect any changes in permanent flags.
Changes made for session flags are only valid for that session.

The currently defined system flags are:

- **Seen**: Message has been read.
- **Answered**: Message has been answered.
- **Flagged**: Message is “flagged” for urgent/special attention.
- **Deleted**: Message is “deleted” for later removal with *IMAP_Delete*, *IMAP_CloseCurrentMB*, *IMAP_SetCurrentMB* or *IMAP_Logout*.
- **Draft**: Message is in draft format; in other words, not complete.
- **Recent**: Message “recently” arrived in this mailbox. This session is the first session notified about this message; subsequent sessions will not see the **¥Recent** flag set for this message. This permanent flag is managed by the IMAP server and cannot be modified by an IMAP client using *IMAP_SetFlags*, for instance.

An IMAP server may allow a client to define new “flags” or, on another IMAP server, may allow managing of flags other than those previously indicated. This depends on the IMAP server implementation. In this case, these special flags are called “keywords” and do not begin with “¥” (see *IMAP_SetCurrentMB*).

Note: If you set the **¥Deleted** flag and close the current session by executing *IMAP_SetCurrentMB*, *IMAP_CloseCurrentMB*, *IMAP_Delete* or *IMAP_Logout*, the message will be “deleted” permanently.

imap_ID is a long integer reference to an open connection created with *IMAP_Login*.

startMsg is a long integer number that specifies the starting message number of the message range to be examined. The message number is a value representing the position of a message within the list of all messages in the current working mailbox.

endMsg is a long integer number that specifies the ending message number of the message range to be examined. The message number is a value representing the position of a message within the list of all messages in the current working mailbox.

Note: The *IMAP_Delete*, *IMAP_MsgLstInfo*, *IMAP_MsgLst*, *IMAP_SetFlags*, *IMAP_GetFlags* and *IMAP_CopyToMB* commands do not return an error if the *startMsg* is greater than the *endMsg*. In the event that this occurs, the command – in effect – does nothing.

msgFlagsList may contain one or several flags. In the case of several flags, the string must be a list of flags, separated by spaces. See examples below.

Only flags listed as *permanentFlags*, (see *IMAP_SetCurrentMB*), will be applied.

deleteOption is an integer value which specifies whether to add or remove the flag(s) specified by the *msgFlagsList* parameter:

- A value of zero will add the flag(s) specified in *msgFlagsList*.
- A value of 1 will remove the flag(s) specified in *msgFlagsList*.

Example 1

Set the **¥Answered** and **¥Draft** flags for the messages specified by *startMsg* and *endMsg*:

```
msgFlagsName:="\Answered \Draft"
 ` \Answered and \Draft are separated by a space (ASCII code)
IMAP_SetFlags(imap_ID;startMsg;endMsg;msgFlagsName;0)
```

Example 2

Remove the **¥Deleted** flag for the messages specified by *startMsg* and *endMsg* whether this flag was set previously or not:

```
msgFlagsName:="\Deleted"
IMAP_SetFlags(imap_ID;startMsg;endMsg;msgFlagsName;1)
```

Example 3

Set the **¥Deleted** flag for the messages specified by *startMsg* and *endMsg* whether this flag was set previously or not:

```
msgFlagsName:="\Deleted"
IMAP_SetFlags(imap_ID;startMsg;endMsg;msgFlagsName;0)
IMAP_CloseCurrentMB(imap_ID)
 `Closes the current mailbox and permanently deletes the specified messages.
```

Example 4

Set the **¥Answered** flag depending on the CheckBoxAnswered value:

```
$Error:=IMAP_SetFlags(vImap_ID;$msgNum;$msgNum;"\Answered";Num(CheckBoxAnswered=0))
```

IMAP_SetPrefs

IMAP_SetPrefs (stripLineFeed ; msgFolder) -> Function result

Parameter	Type		Description
stripLineFeed	Integer	⇒	0 = Do not strip LineFeeds, 1 = Strip LineFeeds, -1 = No Change
msgFolder	Text	⇒	Messages folder path ("" = no change)
Function result	Integer	↩	Error code

Description

The *IMAP_SetPrefs* command sets the preferences for all IMAP commands.

stripLineFeed is an integer value specifying how LineFeed characters will be treated in saved messages. Most IMAP servers combine Carriage Return and Line Feed characters to indicate the end of a line. Macintosh applications prefer a carriage return only as the end-of-line character. This option lets users strip the linefeed character from their message text. A value of zero will leave retrieved messages in the format as stored on the IMAP server. A value of 1 will strip linefeed characters from retrieved messages. A value of -1 will leave this preference as it has been previously set. The default option is set to 1 and will automatically strip linefeeds found in messages.

msgFolder is a text value indicating the local pathname to a folder in which messages retrieved with the *IMAP_Download* command are stored by default.

IMAP_SubscribeMB

IMAP_SubscribeMB (imap_ID ; mbName ; mbSubscribe) -> Function result

Parameter	Type		Description
imap_ID	Longint	→	Reference to an IMAP login
mbName	Text	→	Name of the mailbox to subscribe or unsubscribe
mbSubscribe	Integer	→	0 = Do not subscribe; 1= Subscribe
Function result	Integer	↩	Error code

Description

The *IMAP_SubscribeMB* command allows adding or removing of the specified mailbox name to/from the IMAP server's set of "subscribed" user mailboxes.

As such, the user can choose to narrow down a large list of available mailboxes by subscribing to those that he usually wants to see. To do this, he simply has to use the *IMAP_ListMBs* command with the *subscribedMB* optional parameter set to 1 (see *IMAP_ListMBs*).

imap_ID is a long integer reference to an open connection created with *IMAP_Login*.

mbName is the full name of the mailbox to be subscribed or unsubscribed.

Pass 0 in *mbSubscribe* to not subscribe to the mailbox; pass 1 to subscribe.

IMAP_UIDToMsgNum

IMAP_UIDToMsgNum (imap_ID ; unique_ID ; msgNum) -> Function result

Parameter	Type		Description
imap_ID	Longint	⇨	Reference to an IMAP login
unique_ID	Longint	⇨	Message unique identifier value
msgNum	Longint	⇦	Message number
Function result	Integer	↩	Error code

Description

The *IMAP_UIDToMsgNum* command converts a message *unique_ID* value to its current *msgNum* within the list of messages in the current mailbox referenced by *imap_ID*. Since a specific e-mail message *msgNum* is a floating value relative to other items in the electronic mail list, this command returns the current position of a message whose information may have been retrieved during a prior IMAP session.

imap_ID is a long integer reference to an open connection created with *IMAP_Login*.

unique_ID is a long integer value indicating the **Unique ID** of a message to be located on the IMAP server.

msgNum is a long integer returned containing the current message number (its position within the current message list) of the item identified by *unique_ID*. If the *unique_ID* cannot be found on the server, a zero is returned in *msgNum* and no error is returned.

IMAP_VerifyID

IMAP_VerifyID (imap_ID) -> Function result

Parameter	Type		Description
imap_ID	Longint	→	Reference to this IMAP login
		←	0 = Connection has already closed
Function result	Integer	↩	Error code

Description

An IMAP server will automatically terminate a connection which does not show activity in a period of time determined by its administrator. Each command that interacts with the IMAP server forces a reset of the inactivity timer. The *IMAP_VerifyID* command resets the inactivity timer for the specified IMAP connection without performing any other action. This allows the user to keep a connection active if the possibility exists that the connection may timeout.

When executed, the *IMAP_VerifyID* command will verify the connection has not already been closed. If the connection is still opened, the command will tell the IMAP server to reset the timeout counter for the connection back to zero. If the connection has already closed, *IMAP_VerifyID* will return the appropriate error and free memory used by the IMAP connection, and return a zero value back to *imap_ID*.

imap_ID is a long integer reference to an open connection created with IMAP_Login.

IC Internet

Special Internet Commands, Overview

The set of commands included in this section can be used to perform common tasks over the Internet. Included in this section are commands to 'Ping' and 'Finger' a machine, obtain the time from a time server, resolve a domain name or IP address, and convert a domain name or IP address from or to a long integer. These commands are often used in conjunction with the 4D Internet Commands.

NET_AddrToName

NET_AddrToName (ip_Longint ; hostName ; ip_Address) -> Function result

Parameter	Type		Description
ip_Longint	Longint	→	Long Integer reference to the address
hostName	String	←	Host name
ip_Address	String	←	IP address
Function result	Integer	↩	Error Code

Description

The *NET_AddrToName* command takes a long integer reference to a host name, and returns both the host name and the IP address of that host.

ip_Longint is the long integer reference to an IP address.

hostName is the string returned which contains the host name. If the host name is not able to be resolved, *ip_Address* will return a null string and no error will be returned.

ip_Address is the string returned which contains the IP address.

NET_Finger

NET_Finger (hostName ; searchText ; results) -> Function result

Parameter	Type		Description
hostName	String	→	Host name or IP address
searchText	String	→	Search text
results	Text	←	Finger results
Function result	Integer	↩	Error Code

Description

Given an IP address of a machine to search and the text of a user account name on the machine, the *NET_Finger* command will return the finger results into *results*. The unix finger command is designed to return information about the last login time of a user as well as any additional information the user chooses to provide within his/her ".plan" and ".project" files.

Two different routes may be specified for the Finger search. A finger search may be attempted directly at the user's machine. For instance, to get information about "johnt" at "4d.com", you could perform the search as:

```
$error:=NET_Finger("www.4d.com";"johnt";$fingertext)
```

The same finger search could also be performed indirectly. An indirect search would ask a remote server which supports the finger command to perform your query. For instance, the following will ask the machine identified by the domain name "4d.com" to perform a remote finger query of the user "johnt@4d.com".

```
$error:=NET_Finger("www.4d.com";"johnt@4d.com";$fingertext)
```

While the main information returned in each case should be the same, there are likely to be some subtle differences in the returned text. Different machines may have different options configured when they execute the finger command and the results could vary slightly. Also, there is likely to be some formatting difference between the results of a direct and indirect finger command, with indirect searches often containing additional linefeeds.

hostName is the host name or IP address of the machine in which the user identified by *searchText* has an account.

searchText is either the text to search for on the given finger server, or a machine name or IP address. If *searchText* is a user name, the command will search through the directory of user names on that server for *searchText*. If *searchText* is a machine name or IP address, the command will send a finger request through the finger server in *hostName* to the machine specified.

results is the text returned which contains the results of the search.

NET_NameToAddr

NET_NameToAddr (hostName ; ip_Longint) -> Function result

Parameter	Type		Description
hostName	String	→	Host name or IP address
ip_Longint	Longint	←	Long Integer reference to the address
Function result	Integer	↩	Error Code

Description

The *NET_NameToAddr* command takes a host name or IP address and returns a unique long integer reference to the address.

hostName is the host name or IP address.

ip_Longint is a Longint value representing the IP address specified in the *hostName* parameter. All IP address strings can be converted to a signed Longint value.

While the *ip_Longint* value does not typically have a significant use, some developers may find this command useful to convert IP addresses into a more compact Longint format for data-storage.

NET_Ping

NET_Ping (hostName ; text ; alive ; timeout) -> Function result

Parameter	Type		Description
hostName	String	→	Host name or IP address
text	Text	→	Text to send in ping
alive	Integer	←	1 = Alive, 0 = Timeout/Inactive
timeout	Integer	→	# of seconds to wait, 0 = use IT_SetTimeOut value
Function result	Integer	↩	Error Code

Description

The *NET_Ping* command provides a mechanism to query a remote IP address to see if it is currently active. If the pinged machine is currently running the TCP/IP protocol and the network between the two sites is functional, an 'Alive' status should be returned. Typically, the pinged machine provides no indication to its user of the activity, assuming that the machine can respond to ping (ICMP Echo) requests.

NET_Ping will ping a machine specified by either a host name or IP address. Any machine with an IP address that is accessible via the network can be pinged. This includes end-user machines. [Some security systems known as "firewalls" may hinder you from pinging machines under their protection.]

hostName is the host name or IP address of the machine to ping.

text is the text to send in the ping. The text parameter exists only to effect the size of the TCP packet being sent as the Ping command is executed.

alive is the integer returned corresponding to the state of the machine pinged. A value of 1 indicates the machine is alive. A value of zero indicates the machine is either inactive or the ping timed out before a response was received.

timeout specifies the number of seconds this command will wait for the Ping to complete. This is an optional parameter which, if not supplied, will default to zero. A zero value in this parameter will cause the command timeout if a response is not received by the number of seconds specified with the *IT_SetTimeOut* command.

Note: This parameter is not taken into account under Windows 95/98 and Millennium.

NET_Resolve

NET_Resolve (hostName ; ipOrHost) -> Function result

Parameter	Type		Description
hostName	String	→	Host name or IP address
ipOrHost	String	←	Returns the opposite value
Function result	Integer	↩	Error Code

Description

Given a host name in the first parameter, the *NET_Resolve* command will return the IP address into the second parameter. If the first parameter is passed an IP address, the second parameter will yield the registered host name for that machine.

hostName is a string which contains either an IP address or a host name.

ipOrHost - If the first parameter contained a Host Name then this parameter will receive its IP address. If the IP address was specified in the first parameter then this value will receive its Host Name.

Example

The following example first passes a host name "www.netcom.com" to the **NET_Resolve** command in order to obtain its IP address. The example then makes another call to the command, passing it the IP address in order to obtain its registered host name.

```
C_BOOLEAN($ERR)
C_TEXT($Resolved) //Can be any sized string or text value
$ERR:=ERRCHECK("NET_Resolve";NET_Resolve("www.netcom.com";$Resolved))
 //$Resolved was returned the value '192.100.81.100'
$ERR:=ERRCHECK("NET_Resolve";NET_Resolve($Resolved;$Resolved))
 //$Resolved was returned the value 'www.netcom.com'
```

NET_Time

NET_Time (hostName ; date ; time ; offset) -> Function result

Parameter	Type		Description
hostName	String	→	Host name or IP address
date	Date	←	Date
time	Longint	←	Time, expressed as seconds since midnight
offset	Integer	→	Hours to offset
Function result	Integer	↩	Error Code

Description

Given the host name or IP address to an Internet Time server, the *NET_Time* command will obtain the current date and time from the machine and apply any offset needed to convert to the user's local time.

Note: This command does not affect the computer's internal clock.

hostName is the host name or IP address of an Internet Time server.

date is a 4D Date returned, containing the resulting date after the *offset* has been applied.

time is a LongInt value returned, containing the resulting time after the *offset* has been applied. The value represents the seconds since midnight on *date*. See the example below for a method to convert this value to a 4D Time variable.

offset is the number of hours to add or subtract from the base time values. Internet Time Servers express their values in Universal Time (Greenwich Mean Time). Even if the time server is in your geographic region, it is likely that you will need to supply an *offset* value to compensate for the difference between your local time and Universal time.

Example

The following example obtains the Universal Time from the Time server at "apple.com". The command then subtracts the seven hours specified as the Offset and returns the resulting Date and Time (Time is expressed as a Longint value, which can then be converted using 4D's Time string command, as below).

```
C_DATE(vNetDate)
C_LONGINT(vNetTime)
C_TIME(vTime)
C_LONGINT(vOffset)
If(ERRCHECK("NET_Time";NET_Time("www.apple.com";vNetDate;vNetTime;-7)))
   vTime:=Time(Time string(vNetTime)) //Convert the LongInt time to a 4D Time
End if
```

IC POP3 Review Mail

Receiving Mail, Overview

The suite of POP3 commands enable your database to retrieve messages from a POP3 mail server. 4D Internet Commands are MIME compliant and can recognize and extract messages containing multiple enclosures.

The POP3-related commands are broken down into two sections, "IC POP3 Review Mail" and "IC Downloaded Mail". The separation of commands is representative of the differing methods for reading mail. When reading mail from a POP3 server, messages (or information about the messages) may be brought down into 4D structures (variables, fields, arrays) or they may be downloaded to disk. This section "IC POP3 Review Mail" covers the ability of 4D Internet Commands to read messages from the POP3 server into 4D.

The need to for dual methods of message retrieval is spawned by memory constraints on actions which have the potential to download many megabytes of information. For instance, a single mail message which had a 5 Mb attachment could easily overflow the storage capability within the database. The only 4D structure capable of storing this size is a picture or a BLOB field, but converting a message or attachment to this format is often ineffectual since it places huge memory requirements on any client attempting to access the picture or the BLOB. To resolve this issue, this section has a command *POP3_Download* which will bring a message from the POP3 server to the user's local disk. Once on disk, the "IC Downloaded Mail" section of the manual provides a number of commands to manipulate the file.

When using the suite of POP3 commands, it is important to understand the parameters that are used most frequently, especially *msgNumber* and *uniqueID*. *msgNumber* is the number of a message in the mailbox at the time the *POP3_Login* command was executed. Upon login, messages in a mailbox are assigned numbers from 1 to the number of items in the mailbox. Numbers are assigned based on the order that they were received in the mailbox, with one being the oldest message. The numbers assigned to the messages are only valid during the time from your *POP3_Login* to *POP3_Logout*.

At the time *POP3_Logout* is executed any message marked for deletion will be removed. When the user logs back into the server, the current messages in the mailbox will once again be numbered from 1 to x. For example, if there are 10 messages in the mailbox, and messages numbered 1 through 5 are deleted, messages 6 through 10 will be renumbered 1 through 5 the next time the user logs in to the mailbox.

To illustrate, suppose you login to a POP3 server and obtain the following list of messages:

#	UniqueID	Date	From	Subject
1	bd573a4dbd573a4d	1 Jul 1998 …	jimw@acme.com	Sales lead…
2	bd574dc7bd574dc7	1 Jul 1998 …	frank@acme.com	Site-License order
3	bd575f06bd575f06	3 Jul 1998 …	joe@acme.com	Lunch anyone?
4	bd5761d4bd5761d4	4 Jul 1998 …	kelly@acme.com	Your wife called…
5	bd577dc7db577dc5	4 Jul 1995 …	track@fedex.com	FedEx tracking

During the session you delete message numbers 3 and 4. When you Logout of this session your requested deletions are committed. Now when you log back into the server your message list would be renumbered as:

#	UniqueID	Date	From	Subject
1	bd573a4dbd573a4d	1 Jul 1998 …	jimw@acme.com	Sales lead…
2	bd574dc7bd574dc7	1 Jul 1998 …	frank@acme.com	Site-License order
3	bd577dc7db577dc5	4 Jul 1995 …	track@fedex.com	FedEx tracking

msgNumber is not a static value in relation to any specific message and will change from session to session dependent on its relation to other messages in the mailbox at the time the session was opened. The *uniqueID* however is a unique number assigned to the message when it was received by the server. This number is calculated using the time and date that the message is received and is a value assigned by your POP3 server. Unfortunately, POP3 servers do not use the *uniqueID* as the primary reference to its messages. Throughout the POP3 commands you will need to specify the *msgNumber* as the reference to messages on the server. Developers may need to take some care if developing solutions which bring references to messages into a database but leave the body of the message on the server.

Note: For greater flexibility, 4D Internet commands let you pass a POP3, IMAP or FTP connection reference directly to low-level TCP commands and vice versa. For more information, refer to the **Low Level Routines, Overview** section.

POP3_BoxInfo

POP3_BoxInfo (pop3_ID ; msgCount ; msgSize) -> Function result

Parameter	Type		Description
pop3_ID	Longint	⇒	Reference to a POP3 login
msgCount	Longint	⇐	Number of messages
msgSize	Longint	⇐	Size of all messages
Function result	Integer	↩	Error Code

Description

The *POP3_BoxInfo* command returns information about number and size of messages currently in the mailbox of the open session referenced by *pop3_ID*.

pop3_ID is a long integer reference to an open session created with *POP3_Login*.

msgCount is a long integer value returned containing the number of messages in the mailbox.

msgSize is a long integer value returned containing the total size of all messages in the mailbox.

POP3_Charset

POP3_Charset (decodeHeaders ; bodyCharset) -> Function result

Parameter	Type		Description
decodeHeaders	Integer	⇒	-1 = Use current settings, 0 = Do not manage, 1 = Convert using the Mac OS charset if ISO-8859-1 or ISO-2022-JP, decode extended characters
bodyCharset	Integer	⇒	-1 = Use current settings, 0 = Do not manage, 1 = Convert using the Mac OS charset if ISO-8859-1 or ISO-2022-JP
Function result	Integer	↩	Error Code

Description

The *POP3_Charset* command allows automatic support of messages containing extended characters while processing them with the POP3 and MSG commands. If this command is not called or has parameters set to 0, version 6.7 or higher of 4D Internet Commands will work the same way as version 6.5.x.

POP3_Charset allows first, to set whether the extended characters header decoding has to be managed, and second, whether the message body and headers character set conversions have to be managed.
This command is particularly usefull to support extended characters included in message headers such as "Subject" or mail addresses (for example, to decode an address such as "*=?ISO-8859-1?Q?Test=E9?= <test@n.net >*").

The *decodeHeaders* parameter specifies how to handle header decoding and conversion while executing commands *POP3_MsgLst* or *MSG_FindHeader*(see Compatibility note). Default value is set to 0.

- -1: Use current settings;
- 0: Do not manage;
- 1: Headers are decoded if necessary. If decoded and if the specified character set is ISO-8859-1 or ISO-2022-JP, headers are converted using respectively the Mac OS ASCII code or the Shift-JIS.

Compatibility note (version 6.8.1): *POP3_Charset* applies to *MSG_FindHeader* (in the same way as *POP3_MsgLst*) if the *MSG_Charset* command has not been executed previously.

The *bodyCharset* parameter specifies how to handle message body character set conversion while executing the *MSG_GetBody* command (see Compatibility note). Default value is set to 0.

- -1: Use current settings;
- 0: Do not manage;
- 1: If the "Body-Content-Type" character set is set to ISO-8859-1 or ISO-2022-JP, the message body is converted using respectively the Mac OS ASCII code or the Shift-JIS.

Compatibility note (version 6.8.1): *POP3_Charset* applies to *MSG_GetBody* if the *MSG_Charset* command has not been executed previously.

Example 1

Using version 6.5.x of 4D Internet Commands:

```
$Err:=MSG_FindHeader($msgfile;"From";$from)
$from:=ISO to Mac($from)
$Err:=MSG_FindHeader($msgfile;"To";$to)
$to:=ISO to Mac($to)
$Err:=MSG_FindHeader($msgfile;"Cc";$cc)
$cc:=ISO to Mac($cc)
$Err:=MSG_FindHeader($msgfile;"Subject";$subject)
$subject:=ISO to Mac($subject)

$Err:=MSG_MessageSize($msgfile;$HdrSize;$BdySize;$MsgSize)
$Err:=MSG_GetBody($msgfile;0;$BdySize;$BodyContent)
$BodyContent:=ISO to Mac($BodyContent)
```

Example 2

Using version 6.7 of 4D Internet Commands:

```
$Err:=POP3_Charset(1;1)
$Err:=MSG_FindHeader($msgfile;"From";$from)
$Err:=MSG_FindHeader($msgfile;"To";$to)
$Err:=MSG_FindHeader($msgfile;"Cc";$cc)
$Err:=MSG_FindHeader($msgfile;"Subject";$subject)

$Err:=MSG_MessageSize($msgfile;$HdrSize;$BdySize;$MsgSize)
$Err:=MSG_GetBody($msgfile;0;$BdySize;$BodyContent)
```

Example 3

Using version 6.8 of 4D Internet Commands:

```
$Err:=MSG_Charset(1;1)
$Err:=MSG_FindHeader($msgfile;"From";$from)
$Err:=MSG_FindHeader($msgfile;"To";$to)
$Err:=MSG_FindHeader($msgfile;"Cc";$cc)
$Err:=MSG_FindHeader($msgfile;"Subject";$subject)

$Err:=MSG_MessageSize($msgfile;$HdrSize;$BdySize;$MsgSize)
$Err:=MSG_GetBody($msgfile;0;$BdySize;$BodyContent)
```

POP3_Delete

POP3_Delete (pop3_ID ; startMsg ; endMsg) -> Function result

Parameter	Type		Description
pop3_ID	Longint	→	Reference to a POP3 login
startMsg	Longint	→	Starting message number
endMsg	Longint	→	Ending message number
Function result	Integer	↩	Error Code

Description

Given a range of messages from *startMsg* to *endMsg*, the *POP3_Delete* command will mark each message to be deleted. The act of deleting the message does not occur until you successfully issue the *POP3_Logout* command. If your current session terminates for any reason (timeout, network failure, etc.) prior to calling the *POP3_Logout* command, any messages marked for deletion will remain on the POP3 server.

pop3_ID is a long integer reference to an open session created with *POP3_Login*.

startMsg is a long integer number which is the starting message number of the messages to delete.

endMsg is a long integer number which is the ending message number of the messages to delete.

Note: The *POP3_Delete*, *POP3_MsgLstInfo* and *POP3_MsgLst* commands do not return an error if the *startMsg* is greater than the *endMsg*. In the event that this occurs, this command – in effect – does nothing.

POP3_DownLoad

POP3_DownLoad (pop3_ID ; msgNumber ; headerOnly ; fileName) -> Function result

Parameter	Type		Description
pop3_ID	Longint	→	Reference to a POP3 login
msgNumber	Longint	→	Message number
headerOnly	Integer	→	0 = Entire message, 1 = Only header
fileName	Text	→	Local Filename
		←	Resulting Local Filename
Function result	Integer	↩	Error Code

Description

The *POP3_DownLoad* command is designed to retrieve a message from a POP3 server by downloading it to a disk-based file. Any POP3 message which contains attachments or whose size is greater than 32K should be downloaded with this command. Attachments to a message can only be extracted from messages retrieved in this way.

pop3_ID is a long integer reference to an open session created with *POP3_Login*.

msgNumber is a long integer value indicating which message in the mailbox to retrieve. The *msgNumber* represents the position of a message within the current list of messages. You cannot rely on the *msgNumber* remaining the same for a specific e-mail item from session to session.

headerOnly is an integer value which denotes whether to retrieve the entire contents of the message or just the header information.

fileName contains the name of the file and the optional path where you would like the message saved. This value may be specified three different ways:

""	= Saves the file in the folder set by *POP3_SetPrefs*, with the name "temp1" (if a file with the same name already exists, the filenames "temp2", "temp3", etc. will be tried until an unused file name is found)
"FileName"	= Saves the file in the folder set by *POP3_SetPrefs*, titled *fileName*
"Path:FileName"	= Saves the file in the path specified with the name *fileName*

In the first two cases, if no folder has been specified by *POP3_SetPrefs*, the message will be saved in the same folder as the structure of the database (with 4D single-user) or in the 4D Client folder (with 4D Server). After the file has been saved to disk, the final name of the file will be returned to the variable passed as the *fileName* parameter. If you attempt to call *POP3_Download* with a *fileName* that already exists within the download folder, the name will be numerically incremented and its new value as saved to disk will be returned to the *fileName* variable.

POP3_GetMessage

POP3_GetMessage (pop3_ID ; msgNumber ; offset ; length ; msgText) -> Function result

Parameter	Type		Description
pop3_ID	Longint	⇒	Reference to a POP3 login
msgNumber	Longint	⇒	Message number
offset	Longint	⇒	Offset of character at which to begin retrieval
length	Longint	⇒	How many characters to return
msgText	Text	⇐	Message Text
Function result	Integer	↩	Error Code

Description

The *POP3_GetMessage* command returns the complete text of the message identified by *msgNumber* within the mailbox referenced by *pop3_ID*. Unless otherwise specified by the *POP3_SetPrefs* command, any linefeed characters within the message will be removed. The *POP3_GetMessage* command returns the entire block of the message, including header information.

pop3_ID is a long integer reference to an open session created with *POP3_Login*.

msgNumber is a long integer value indicating which message in the mailbox to retrieve. The *msgNumber* represents the position of a message within the current list of messages. You cannot rely on the *msgNumber* remaining the same for a specific e-mail item from session to session.

offset is a long integer value indicating the number of characters from the beginning of the message to begin reading. In most circumstances a zero should be passed to this parameter.

length is a long integer value representing the number of characters beyond the *offset* position to retrieve. Since the maximum length of this parameter is limited to 32,000 characters for historical reasons, the *length* parameter should be set to any number below 32,000. Messages whose size is greater than 32K must be retrieved to disk via the *POP3_Download* command.

msgText is a text variable which will receive the retrieved text.

POP3_GetPrefs

POP3_GetPrefs (stripLineFeed ; msgFolder ; attachFolder) -> Function result

Parameter	Type		Description
stripLineFeed	Integer	←	0 = Don't Strip LF, 1 = Strip LF
msgFolder	Text	←	Messages folder path ("" = no change)
attachFolder	Text	←	Attachments folder path ("" = no change)
Function result	Integer	↩	Error Code

Description

The *POP3_GetPrefs* command returns the current preferences for the POP3 commands. The preferences are returned into the variables listed in the parameters.

stripLineFeed returns the current setting of the users preference for linefeed stripping.

msgFolder is a text variable which returns the local pathname to the default folder in which retrieved messages are stored.

attachFolder is a text variable which returns the local pathname to the default folder in which extracted attachments are stored.

Compatibility note (version 6.8.1): The *attachFolder* parameter of the *POP3_GetPrefs* command is optional; therefore, we recommend that you do not pass this parameter since it will no longer be used. Note that this parameter does not affect POP3 commands since it is only used by MSG commands.

POP3_Login

POP3_Login (hostName ; userName ; password ; aPOP ; pop3_ID {; sessionParam}) -> Function result

Parameter	Type		Description
hostName	String	→	Host Name or IP address of the POP3 mail server
userName	String	→	User name
password	String	→	Password
aPOP	Integer	→	0 = Cleartext Login, 1 = APOP Login
pop3_ID	Longint	←	Reference to this POP3 login
sessionParam	Longint	→	1 = Use SSL, 0 or omitted = Do not use SSL
Function result	Integer	↩	Error Code

Description

The *POP3_Login* command logs the user into the POP3 mail server with the given *userName* and *password*. If *aPOP* is 1 then the APOP mechanism (RFC#1321) is used to login. If *aPOP* is zero or not given then a normal cleartext password login is performed. The particular login is given a reference (*pop3_ID*) which subsequent commands can refer to.

Warning: POP3 servers were not designed to be accessed in an interactive fashion. Once you have logged in to a server you should perform whatever actions are needed and then log out of the server as soon as possible. Between your calls of *POP3_Login* and *POP3_Logout*, your procedure should not sit in any user-interactive screen. A POP3 server will automatically disconnect any sessions which do not show activity for a certain period of time. According to the RFC for POP3, the inactivity timer is supposed to be a minimum of 30 minutes. However, our experience has shown that most servers force inactive clients out after a much shorter period of time.

Each command that interacts with the POP3 server will force a reset of your inactivity timer. In the event that the server aborts your connection before you have issued a *POP3_Logout* call, any deletions you had performed would be rolled back.

hostName is the host name or IP address of the POP3 mail server. It is recommended that the host name be used, but if needed an IP address can be used.

userName is the user's name on the POP3 mail server. The *userName* should not contain the domain. For example, for the address "jack@4d.com", *userName* would be just "jack".

password is the password for *userName* on the POP3 mail server.

aPOP is an integer value indicating whether the APOP mechanism is used to login. A value of 1 will use the APOP mechanism. A zero value will perform a cleartext password login. The default value is zero.

pop3_ID is a long integer variable into which is returned a reference to the session just established. The variable will be used in all subsequent commands which perform actions related to this session.

The optional *sessionParam* parameter enables the SSL protocol for a connection:

- If you pass 1, the connection to the POP3 server will be made in SSL (synchronous mode),
- If you pass 0 or omit this parameter, the connection will be made in standard, non-secure mode.

POP3_Logout

POP3_Logout (pop3_ID) -> Function result

Parameter	Type		Description
pop3_ID	Longint	⇒	Reference to a POP3 login
		⇐	0 = Command successfully logs off
Function result	Integer	↩	Error Code

Description

The *POP3_Logout* command will log out of the open POP3 session referred to by the *pop3_ID* variable. If the command successfully logs off the POP3 server a zero value is returned back as the current *pop3_ID*.

Logging out from a POP3 server will signal the server that you wish to commit any deletions you made during that session. To rollback any deletions you may have made prior to logout, use the *POP3_Reset* command prior to *POP3_Logout*.

pop3_ID is a long integer reference to an open session created with *POP3_Login*.

POP3_MsgInfo

POP3_MsgInfo (pop3_ID ; msgNumber ; msgSize ; uniqueID) -> Function result

Parameter	Type		Description
pop3_ID	Longint	→	Reference to a POP3 login
msgNumber	Longint	→	Message number
msgSize	Longint	←	Message size
uniqueID	String	←	Unique ID of message on server
Function result	Integer	↩	Error Code

Description

The *POP3_MsgInfo* command returns information about the message identified by *msgNumber* within the open mailbox referenced by *pop3_ID*. Information about the size of the message and its Unique ID will be returned.

pop3_ID is a long integer reference to an open session created with *POP3_Login*.

msgNumber is a long integer value indicating which message in the mailbox you wish to retrieve information about. The *msgNumber* represents the position of a message within the current list of messages. You cannot rely on the *msgNumber* remaining the same for a specific e-mail item from session to session.

msgSize is the long integer value returned containing the size of the message referenced by *msgNumber*.

uniqueID is a string variable denoting the Unique ID of the message on the server. The *uniqueID* is a value assigned to the message by the POP3 server software. This value will not change from session to session in the same way as *msgNumber*. The *uniqueID* value is a good reference to verify if your database has already downloaded a message from the server.

POP3_MsgLst

POP3_MsgLst (pop3_ID ; start ; end ; hdrArray ; msgNumArray ; idArray ; valueArray) -> Function result

Parameter	Type		Description
pop3_ID	Longint	→	Reference to a POP3 login
start	Longint	→	Start message number
end	Longint	→	End message number
hdrArray	String array	→	Array of Headers to retrieve
msgNumArray	Longint array	←	Array of message numbers
idArray	String array	←	String array of Unique ID's
valueArray	2D String array, 2D Text array	←	2D Array of header values
Function result	Integer	↩	Error Code

Description

The *POP3_MsgLst* command is used to get specific information of mailbox contents. *hdrArray* is a string or text array which lists the specific mail headers you wish to retrieve. *valueArray* is a 2-dimensional array which receives the data for each header specified in *hdrArray*. Each requested header will have a corresponding array in the first dimension of *valueArray*.

This command allows the user to request specific columns of the message list. This command can only return values of header items, it cannot be used to retrieve the body of a message.

Note: Since mail headers can include extended characters, you can automate their management using the *POP3_Charset* command.

Example

```
aHeaders{1}:="Date:"
aHeaders{2}:="From:"
aHeaders{3}:="Subject:"
POP3_MsgLst(◊POP3_ID;vStart;vEnd;aHeaders;aMsgNum;aUIDs;aValues)
aValues{1}{1}may equal"Thu, 19 November 1998 00:24:02 -0800"
aValues{2}{1}may equal"Jack@4d.com"
aValues{3}{1}may equal"Call your wife"
```

Errors are handled in the following manner:

1) Only communication-related error codes will be returned. If the command can't complete its task because of an error (network, syntax, server, etc.) then the appropriate error code will be returned.

2) If a message within the specified range of messages does not exist or gets an error:
-- No array element is created for that message.
-- No error code will be returned

3) The inability to locate any or all of the specified headers within any message does not constitute an error:
-- An array element for the message will be created
-- The Message Number and UniqueID array element will contain the appropriate values
-- For each header which does not exist in the message, a null string will be returned into that array element
-- No error code will be returned

Note: The *POP3_Delete*, *POP3_MsgLstInfo* and *POP3_MsgLst* commands do not return an error if the *startMsg* is greater than the *endMsg*. In the event that this occurs, this command – in effect – does nothing.

POP3_MsgLstInfo

POP3_MsgLstInfo (pop3_ID ; startMsg ; endMsg ; sizeArray ; msgNumArray ; idArray) -> Function result

Parameter	Type		Description
pop3_ID	Longint	⇒	Reference to a POP3 login
startMsg	Longint	⇒	Start message number
endMsg	Longint	⇒	End message number
sizeArray	Longint array	⇐	Array of sizes
msgNumArray	Longint array	⇐	Array of message numbers
idArray	String array	⇐	Array of Unique ID's
Function result	Integer	↩	Error Code

Description

The *POP3_MsgLstInfo* command returns information about a set of messages in a mailbox. The information is returned into three arrays with each element of the arrays corresponding to one message. Information is returned about the size of each message, the message number, and the Unique-ID of the message. The arrays passed as parameters must be of pre-declared types, though they may be of any size. The *POP3_MsgLstInfo* command will reset the size of each array to number of messages retrieved.

The *POP3_MsgLstInfo* command will not return an error number if it fails to retrieve information on any message within the current message list. If an error is encountered, no element is created in the arrays for the problem message. If the command reads each message successfully, the *msgNumArray* should contain numeric values in a sequential order. If problems were encountered, there may be gaps in the sequence of numbers held in *msgNumArray*.

pop3_ID is a long integer reference to an open session created with *POP3_Login*.

startMsg is a long integer number which specifies the starting message number of the message range to be examined. The message number is a value representing the position of a message within the list of all messages in the mailbox identified by *pop3_ID*.

endMsg is a long integer number which specifies the ending message number of the message range to be examined. The message number is a value representing the position of a message within the list of all messages in the mailbox identified by *pop3_ID*.

sizeArray is a long integer array returned containing the sizes of each message between *startMsg* and *endMsg*.

msgNumArray is a long integer array returned containing the message numbers between *startMsg* and *endMsg*.

idArray is a string or text array returned containing the Unique-ID's of the messages between *startMsg* and *endMsg*.

Note: The *POP3_Delete*, *POP3_MsgLstInfo* and *POP3_MsgLst* commands do not return an error if the *startMsg* is greater than the *endMsg*. In the event that this occurs, this command – in effect – does nothing.

POP3_Reset

POP3_Reset (pop3_ID) -> Function result

Parameter	Type		Description
pop3_ID	Longint	⇒	Reference to a POP3 login
Function result	Integer	⮌	Error Code

Description

The *POP3_Reset* command resets the high message count and undeletes any messages marked as deleted during the current session.

Note: The *POP3_Delete* command only sets a flag for messages to be deleted. Messages on a POP3 server are only deleted at the time of a successful logout (*POP3_Logout*).

pop3_ID is a long integer reference to an open session created with *POP3_Login*.

POP3_SetPrefs

POP3_SetPrefs (stripLineFeed ; msgFolder ; attachFolder) -> Function result

Parameter	Type		Description
stripLineFeed	Integer	⇒	0 = Don't Strip LineFeeds, 1 = Strip LineFeeds, -1 = No Change
msgFolder	Text	⇒	Messages folder path ("" = no change)
attachFolder	Text	⇒	Attachments folder path ("" = no change)
Function result	Integer	↩	Error Code

Description

The *POP3_SetPrefs* command sets the preferences for all POP3 commands.

stripLineFeed is an integer value specifying how LineFeed characters will be treated in saved messages. Most POP3 servers combine Carriage Return and Line Feed characters to indicate the end of a line. Macintosh applications prefer a carriage return only as the end-of-line character. This option lets users strip the linefeed character from their message text. A value of zero will leave retrieved messages in the format as stored on the POP3 server. A value of 1 will strip linefeed characters from retrieved messages. A value of -1 will leave this preference as it has been previously set. The default option defaults to 1 and will automatically strip linefeeds found in messsages.

msgFolder is a text value indicating the local pathname to a folder in which messages retrieved with the *POP3_Download* command are stored by default.

Compatibility note (version 6.8.1): the *stripLineFeed* and *msgFolder* parameters were previously applied to MSG_Commands. This is no longer the case when the *MSG_SetPrefs* command is used.

attachFolder is a text value containing the local pathname to a folder in which attachments are stored when the *MSG_Extract* command separates the attachments from the main body of a message.

Compatibility note (version 6.8.1): the *attachFolder* parameter is also found in *POP3_SetPrefs* and *MSG_SetPrefs* therefore you can modify it using either of these two commands. Using the *MSG_SetPrefs* command is strongly recommended; the *POP3_SetPrefs* parameter, used for compatibility reasons, will not be used in the future. The *attachFolder* of the *POP3_SetPrefs* command is optional therefore we recommend that you do not pass this parameter. This recommendation also applies to *POP3_GetPrefs*.

POP3_UIDToNum

POP3_UIDToNum (pop3_ID ; uniqueID ; msgNumber) -> Function result

Parameter	Type		Description
pop3_ID	Longint	⇒	Reference to a POP3 login
uniqueID	String	⇒	Unique ID of message on server
msgNumber	Longint	⇐	Message number
Function result	Integer	↩	Error Code

Description

The *POP3_UIDToNum* command converts a message's Unique ID value to its **current** *msgNumber* within the list of messages in the mailbox referenced by *pop3_ID*. Since a specific mail message's *msgNumber* is a floating value relative to other items in the mail list, this command returns the current position of a message whose information may have been retrieved during a prior POP3 session.

pop3_ID is a long integer reference to an open session created with *POP3_Login*.

uniqueID is a string value containing the Unique-ID of a message to locate on the POP3 server. This command will look for this value in the message headers of the account referenced by *pop3_ID*. Once found, the message's current position in the listing will be returned in *msgNumber*.

msgNumber is a long integer returned containing the current message number (its position within the current message list) of the item identified by *uniqueID*. If the *uniqueID* cannot be found on the server, a zero is returned in *msgNumber* and no error is returned.

POP3_VerifyID

POP3_VerifyID (pop3_ID) -> Function result

Parameter	Type		Description
pop3_ID	Longint	→	Reference to a POP3 login
		←	0 = Connection has already closed
Function result	Integer	↩	Error Code

Description

A POP3 server will automatically terminate sessions which do not show activity in a period of time determined by its administrator. Each command that interacts with the POP3 server forces a reset of the inactivity timer. The *POP3_VerifyID* command resets the inactivity time for the specified POP3 session without performing any other action. This allows the user to keep a session active if the possibility exists that the session may timeout.

When executed, the *POP3_VerifyID* command will verify the connection has not already been closed. If the session is still open the command will tell the POP3 server to reset the timeout counter for the session back to zero. If the connection has already closed, *POP3_VerifyID* will return the appropriate error and free memory used by the POP3 session, and return a zero value back to *pop3_ID*.

pop3_ID is a long integer reference to an open session created with *POP3_Login*.

IC Send Mail

Sending Mail, Overview

Simple Mail Transport Protocol (SMTP) is the mail standard used throughout the Internet. With 4D Internet Commands, developers can build simple mail messages with just one command, or complex messages with a series of commands. The SMTP commands enable developers to control all portions of a mail message, including Reply-To headers, Sender headers, Attachments, Comments, and References.

4D and 4D Internet Commands allow developers to create very powerful databases with the ability to send messages and attachments over the Internet. Some examples of how the suite of SMTP commands could enhance your databases are:

- Automation of sending reports or documents created within 4D.
- Databases could inform developers of special occurrences (i.e. ON ERR CALL("Mail_Error"))
- Databases could execute automated mailings to people across the country

There are an unlimited number of uses for the suite of SMTP commands. These commands, combined with those for POP3 (retrieving both files and attachments), FTP, and TCP provide the 4D developer with the tools to dramatically increase the communications capabilities of their 4D databases.

Two methods of Creating a Mail Message

Within the SMTP section of commands, there are two separate methods of sending electronic mail, which have previously been referred to as "simple" and "complex". The "simple" method involves a single command, *SMTP_QuickSend*, which accepts all the parameters necessary to address and send a message.

The majority of e-mail sent throughout the world is pretty simple in its construction; somebody "here" wants to send a "message" of some kind to somebody "there" regarding some "subject". If this were a paper letter, you would write everything up, seal and address the envelope and then take it to the post office for delivery. With SMTP_QuickSend, you can specify the "From", "To", "Subject" and "Message Body" within one command for easy e-mail delivery.

However, not all mail delivery can fit into such narrow parameters. For instance, suppose the letter above needed copies sent to other interested parties or perhaps an attachment such as your Annual Report needed to be enclosed. In these cases, photocopies of your letter would be made and reports printed as your staff collated the material and addressed the envelopes to each recipient. The SMTP commands in 4D Internet Commands simplify the electronic distribution by giving you control over all aspects of e-mail delivery. Multiple attachments, Carbon Copy & Blind Carbon Copy addressing, any mail header specification can be handled through the Built Message capabilities of the SMTP commands.

Understanding Mail Delivery

One of the critical concepts in understanding the SMTP commands relates to the method in which mail is delivered to its recipients. The SMTP commands do not directly deliver the mail to each recipient. The commands handle the proper composition and formatting of your mail and will deliver the results to the SMTP server specified by the SMTP_Host command. The SMTP server is often a machine within your own organization or at your Internet service provider. The SMTP server then resolves the optimum delivery path for your mail and schedules its distribution based on settings configured by the mail administrator.

Minimum Requirements to Send a complex SMTP Message

In order to successfully deliver a mail message built using the SMTP commands, the essential commands must all be correctly defined. The following commands represent the minimum in order for e-mail delivery to be successful:

- *SMTP_New*
  Creates the space in memory for the new message and returns a reference to be used in subsequent

commands.

- *SMTP_Host*
  Specifies the SMTP server where the message will be delivered
- *SMTP_From*
  At least one address in this header
- *SMTP_To*
  At least one address in this header
- *SMTP_Send*
  Sends the message
- *SMTP_Clear*
  Clears any memory used during the creation of the message

If only the commands listed above were executed, a message would have been sent which contained no "Subject" definition and no message body. This isn't particularly useful and illustrates the need to specify additional detail in order to effectively communicate your message.

Unicode by default (4D v14)

Starting with 4D v14, the "subject" and "body" fields of SMTP commands use the UTF-8 character set by default. This character set will be interpreted correctly by almost all of the e-mail clients. This functioning greatly simplifies the use of SMTP commands and now limits the usefulness of the **SMTP_Charset** and **SMTP_SetPrefs** commands.

SMTP_AddHeader

SMTP_AddHeader (smtp_ID ; headerName ; headerText {; deleteOption}) -> Function result

Parameter	Type		Description
smtp_ID	Longint	→	Message reference
headerName	String	→	Name of header
headerText	Text	→	Header text
deleteOption	Integer	→	0 = Add 1 = Replace all headers with 'headerName', 2 = Remove all headers named 'headerName'
Function result	Integer	←	Error Code

Description

The *SMTP_AddHeader* command allows users to add their own headers to the message referenced by *smtp_ID*. Beyond the various headers 4D Internet Commands have provided commands for, there are two additional categories of headers, these being 'user-defined' and 'extended' headers. The *SMTP_AddHeader* command permits the user to add both the new header tag and the data to associate with it.

Extended-Headers: These headers have been officially recognized by the NIC and were defined after the original SMTP specification. These headers often have a specific function to effect behavior in various software applications. Extended headers never begin with the letter "X".

User-Defined Headers: The SMTP protocol allows anyone to create their own header definitions. All user-defined headers should begin with the characters "X-" so there will be no possibility of conflict with a future Extended-Header. User-defined headers are tremendously useful when you have design control over both ends of the communications.

User defined headers allow the developer to store data which can easily be pulled out using the POP3 external command *MSG_FindHeader*. For example, you may create a header named "X-001001", which contains the value in field 01 of file 01. An unlimited number of headers may be added to a message. User-defined headers give the user the ability to add information that can easily be extracted without the need to parse through the body of the message to find the appropriate information.

smtp_ID is the long integer reference to the mail message created with the *SMTP_New* command.

headerName is a string which contains the name of the header to be added.

headerText is a text value containing the information to be assigned to the field identified by *headerName*.

Warning: The text should not contain a line feed (ascii=10). Doing so would signify the end of the header section and the beginning of the body. Subsequent header items could be pushed into the body and not recognized properly by the server or client software. For more information regarding the headers, please refer to RFC#822.

Note: The command does nothing if *headerName* or *headerText* is an empty string (no header is added).

deleteOption is an optional integer parameter which specifies whether to delete the current header. A value of zero will add *headerName* to the message. A value of 1 will replace all headers with *headerName*. In this case, if *headerName* is a null string, all headers will be removed. A value of 2 will remove all headers named *headerName*.

Note: Beginning with version 14 of 4D Internet Commands, when you want to send an HTML message, you no longer need to change the "Content-Type" header using **SMTP_AddHeader**. You can declare the use of the HTML format directly using the **SMTP_Body** command, in which case the "Content-Type" will be defined automatically as "text/html;charset=utf-8" (otherwise, the "Content-Type" is set by default as "text/plain;charset=utf-8"). However, for specific needs, you can always "force" the "Content-Type" field with **SMTP_AddHeader**. In this case, make sure to specify the charset (normally "charset=utf-8" since, by default, 4D IC always sends the body as UTF-8).

SMTP_Attachment

SMTP_Attachment (smtp_ID ; fileName ; encodeType ; deleteOption {; attachmentID {; contentType}}) -> Function result

Parameter	Type		Description
smtp_ID	Longint	→	Message reference
fileName	Text	→	Name of file to attach
encodeType	Integer	→	0 = No encoding (sends DataFork only) ±1 = BinHex ±2 = Base64; (sends DataFork only) ±3 = AppleSingle ±4 = AppleDouble ±5 = AppleSingle AND Base64 ±6 = AppleDouble AND Base64 ±7 = UUEncode
deleteOption	Integer	→	0 = Add to existing list, 1 = Replace all attachments with Filename, 2 = Remove only this attachment
attachmentID	Text	→	ID of attachment (HTML messages only)
contentType	Text	→	Value of content type to set
Function result	Integer	↩	Error Code

Description

The **SMTP_Attachment** command provides a means to attach text or binary files to your message in MIME format. This command may be called multiple times in order to attach multiple documents to one mail message. If a value greater than zero is passed to the *encodeType* parameter, this command will perform encoding at the time the message is sent.

smtp_ID is the long integer reference to the mail message created with the *SMTP_New* command.

fileName contains the name of the file you want to attach to the message. This value may be specified three different ways:

""	= Display the Standard Open File dialog.
"FileName"	= Looks for FileName in the same directory as the structure of the database.
"Path:FileName"	= Complete path of the file including FileName.

encodeType is an Integer value indicating what type of encoding will be done on the file before it is incorporated into the message. If attaching a binary file, an encoding method must be applied capable of the proper conversion (BinHex, AppleSingle). The most common encoding method is BinHex.

If you pass positive values of *encodeType* the command will automatically encode the file using the specified method when the message is sent. The encoding of a file occurs at the time the *SMTP_Send* command is issued. If the file is large it may take some time for the *SMTP_Send* command to complete. Significant time may be saved in cases where the same file will be sent a number of times. In these cases it is best to encode the file one time with the *IT_Encode* command and then attach the resulting file to your message using the negative value of *encodeType*. A negative value in *encodeType* will not perform any additional encoding but will set the message headers to the correct encoding method of the attached file. This will inform your recipients' mail reader of the correct way to interpret your attachment.

Note: You cannot pass an array element to the *encodeType* parameter.

deleteOption is an optional integer parameter which specifies how to treat the attachment.

- A value of zero will add the attachment to the current list of attachments.
- A value of 1 will replace all attachments with the file in *fileName*. If *fileName* is a null string, all attachments will be removed.
- A value of 2 will remove only the attachment listed in *fileName* from the list of attachments.

The *attachmentID* parameter associates the attachment with a reference defined in the message body using the HTML tag **<img src=¥"cid:ID¥">**. This means that the contents of the files, for example a picture, can be displayed within the message on the e-mail client.
This functioning is only supported with messages in HTML. Also note that the final result may vary depending on the e-mail client.

The optional *contentType* parameter explicitly sets the content type of the attachment file. By default, if this parameter is omitted or contains an empty string, 4DIC automatically sets the content type of the attachment file based on its extension. The following rules are applied:

Extension	Content Type
jpg, jpeg	image/jpeg
png	image/png
gif	image/gif
pdf	application/pdf
doc	application/msword
xls	application/vnd.ms-excel
ppt	application/vnd.ms-powerpoint
zip	application/zip
gz	application/gzip
json	application/json
js	application/javascript
ps	application/postscript
xml	application/xml
htm, html	text/html
mp3	audio/mpeg
other	application/octet-stream

In *contentType*, you can pass a string defining the content type (MIME type) for the file, for example "video/mpeg". This content-type value will be set for the attachment, regardless of its extension.

Note: Pass an empty string ("") in *attachmentID* if you do not want to use this parameter.

Example 1

Sending an HTML message with a picture included:

```
 $error:=SMTP_New($smtp_id)
 $error:=SMTP_Host($smtp_id;"smtp.gmail.com")
 $error:=SMTP_From($smtp_id;"henry@gmail.com")
 $error:=SMTP_ReplyTo($smtp_id;"replies@gmail.com")
 $error:=SMTP_Subject($smtp_id;"HTML Test  &  picture included")
 $error:=SMTP_To($smtp_id;"john@4d.com";1)
 $error:=SMTP_Body($smtp_id;"<html><B><I>Hello world in bold!</I></B> <img
src=\"cid:myID123\">(normal text)</HTML>";4)
 $error:=SMTP_Attachment($smtp_id;"c:\\temp\\tulips.jpg";2;0;"myID123")
 $error:=SMTP_Auth($smtp_id;"henry@gmail.com";"*******")
 $error:=SMTP_Send($smtp_id;1)
 $error:=SMTP_Clear($smtp_id)
```

Example 2

You want to declare your settings files as XML files

```
 $path:=Get 4D folder(Database folder)+"Settings.mySettings"
 $err:=SMTP_Attachment($smtp_id;$path;2;0;"myID123";"application/xml")
```

SMTP_Auth

SMTP_Auth (smtp_ID ; userName ; password {; authMode}) -> Function result

Parameter	Type		Description
smtp_ID	Longint	⇨	Message reference
userName	String	⇨	User name to be used for SMTP authentication
password	String	⇨	Password to be used for SMTP authentication
authMode	Integer	⇨	Authentication mode to be used: 0 or omitted = Mode defined by server 1= PLAIN, 2 = LOGIN, 3 = CRAM-MD5
Function result	Integer	⇦	Error code

Description

The *SMTP_Auth* command allows sending a message referenced by *smtp_ID* when an authentication mechanism is required by the SMTP server. This type of authentication is required by some SMTP servers in order to reduce the risk that messages have been falsified or that the sender's identity has been usurped, in particular for the purpose of spamming.

This command can be used whether authentication is needed or not since it is only executed if *userName* and *password* are not null strings.

smtp_ID is the long integer reference to the mail message created with the *SMTP_New* command.

userName is the authentication user name on the SMTP server. *userName* should not contain the domain. For example, for the address “jack@4d.com,” *userName* would just be “jack.”

password is the authentication password for *userName* on the SMTP server.

Note: If *userName* and/or *password* are null strings, the *SMTP_Auth* command is not executed.

The optional *authMode* parameter allows the “forcing” of the authentication mode used. You can pass 0, 1, 2 or 3 in this parameter:

- If you pass 0 (zero), the authentication mode used by the *SMTP_Auth* command will be the most secure mode supported by the server (CRAM-MD5, LOGIN then PLAIN),
- If you pass 1, the authentication method used will be PLAIN,
- If you pass 2, the authentication method used will be LOGIN,
- If you pass 3, the authentication method used will be CRAM-MD5.
  If *authMode* is omitted, the value 0 is used by default. If the authentication method requested by this parameter is not supported by the SMTP server, an error is returned.

Example

This example enables sending a message with or without authentication depending on the content of specific fields stored in the 4D database:

```
C_LONGINT($vError)
C_LONGINT($vSmtp_id)
C_TEXT($vAuthUserName;$vAuthPassword)

$vError:=SMTP_New($vSmtp_id)
$vError:=SMTP_Host($vSmtp_id;"wkrp.com")
$vError:=SMTP_From($vSmtp_id;"herb_tarlick@wkrp.com")
$vError:=SMTP_Subject($vSmtp_id;"Are you there?")
$vError:=SMTP_To($vSmtp_id;"Dupont@wkrp.com")
$vError:=SMTP_Body($vSmtp_id;"Can we have a meeting?")

 //The fields are entered if the server uses an authentication mechanism.
 //Otherwise, null strings are returned.
$vAuthUserName:=[Account]AuthUser
$vAuthPassword:=[Account]AuthPass
```

```
$vError:=SMTP_Auth($vSmtp_id;$vAuthUserName;$vAuthPassword)
$vError:=SMTP_Send($vSmtp_id)
$vError:=SMTP_Clear($vSmtp_id)
```

SMTP_Bcc

SMTP_Bcc (smtp_ID ; blindCarbon {; deleteOption}) -> Function result

Parameter	Type		Description
smtp_ID	Longint	⇒	Message reference
blindCarbon	Text	⇒	AddressList
deleteOption	Integer	⇒	0 = Add, 1 = Replace, 2 = Delete
Function result	Integer	↩	Error Code

Description

The *SMTP_Bcc* command adds blind carbon copy recipients to the message specified by *smtp_ID*. It is not mandatory to have any addresses in the Bcc: field.

The only way to keep AddressList information confidential when sending mail to groups of people is to list the addresses within the "Bcc" header. The addresses listing in a "Bcc" header are not sent as part of the message header or body. The addresses will not be viewable by any recipient specified in the "To", "Cc" or "Bcc" headers.

The "Bcc" recipients will be able to see all "To" and "Cc" recipients, but they will not be able to see other "Bcc" recipients. Often group mailings to a large number of recipients will be addressed with all recipients placed in the "Bcc" header. This prevents the users from having large address lists cluttering the message and keeps them from accessing the addresses of others.

Another reason for the use of "Bcc" is that many mail applications have a "Reply All" feature which will add all recipients in the "To" and "Cc" sections to the replying message. Placing all recipient addresses within the "Bcc" header will prevent users from replying to every person who received the original message.

smtp_ID is the long integer reference to a message created with the *SMTP_New* command.

blindCarbon is a text value containing an AddressList of one or more mail addresses.

deleteOption is an integer value which specifies whether to add or delete the "Bcc" header:

- A value of zero will add the new value to the "Bcc" field.
- A value of 1 will set the "Bcc" field to the new value, overriding any prior settings (if you pass an empty string in *blindCarbon*, the header will be removed from the mail envelope).
- A value of 2 will delete any address previously defined for the "Bcc" field and remove the header from the mail envelope.
  deleteOption is an optional parameter which will default to zero if not otherwise specified.

Example

In this example a message is created and the static elements are defined outside the scope of the 'for' loop. Then, for each record in the [People] table, an address is added to the blind carbon copy list.

```
$error:=SMTP_From($smtp_id;"sales@massmarket.com")
$error:=SMTP_Subject($smtp_id;"Terrific Sale! This week only!")
$error:=SMTP_Body($smtp_id;$GenericBody)
For($i;1;Records in selection([People]))
   $error:=SMTP_Bcc($smtp_id;[People]Email;0) `Add this email address to the BCC list
   NEXT RECORD([People])
End for
$error:=SMTP_Send($smtp_id) `Send the message to everyone
$error:=SMTP_Clear($smtp_id)
```

SMTP_Body

SMTP_Body (smtp_ID ; msgBody {; option}) -> Function result

Parameter	Type		Description
smtp_ID	Longint	⇒	Message reference
msgBody	Text	⇒	Body of message
option	Integer	⇒	0 = Replace (if msgBody not empty), 1 = Delete, 2 = Append, 4 = HTML text (plain text by default)
Function result	Integer	⇐	Error Code

Description

The **SMTP_Body** command assigns the text in *msgBody* to the main body section of the mail message identified by *smtp_ID*. The *msgBody* is the main block of text.

smtp_ID is the long integer reference to the mail message created with the **SMTP_New** command.

msgBody is a text value which contains the body of the message. By default, the body of the message is encoded with the UTF-8, which ensures that the characters sent will be interpreted correctly by almost all of the e-mail clients. If you want to use a specific character set, refer to the **SMTP_SetPrefs** and **SMTP_Charset** commands.

option is used to manage the concatenation of several bodies and to modify the message format (text or HTML):

- A value of zero will set the body to the provided text string, overriding any prior settings (if you pass an empty string in *msgBody*, the prior text will be used).
- A value of 1 will set the body to the provided text string, overriding any prior settings (if you pass an empty string in *msgBody*, the body of the message will be deleted).
- A value of 2 will append the text in *msgBody* to the body of the message, after any text that had already been defined (concatenation).
- A value of 4 indicates that the contents of *msgBody* is HTML (by default, *msgBody* is sent as plain text).
  If *option* is omitted, by default the value 0 is used.

To combine sending the message in HTML with a replacement option, you can just add together the values. For example, you can pass 1+4 in *option* to replace the body and send the message in HTML.

Example

Here is a complete SMTP example:

```
C_LONGINT($SMTP_ID)
C_BOOLEAN($SentOK;$OK)
$SentOK:=False `A flag to indicate if we made it through all of the commands
Case of
   :(Not(ERRCHECK("SMTP_New";SMTP_New($SMTP_ID))))
   :(Not(ERRCHECK("SMTP_Host";SMTP_Host($SMTP_ID;◊pref_Server))))
   :(Not(ERRCHECK("SMTP_From";SMTP_From($SMTP_ID;vFrom))))
   :(Not(ERRCHECK("SMTP_To";SMTP_To($SMTP_ID;vTo))))
   :(Not(ERRCHECK("SMTP_Cc";SMTP_Cc($SMTP_ID;vCC))))
   :(Not(ERRCHECK("SMTP_Bcc";SMTP_Bcc($SMTP_ID;vBcc))))
   :(Not(ERRCHECK("SMTP_Subject";SMTP_Subject($SMTP_ID;vSubject))))
   :(Not(ERRCHECK("SMTP_Comments";SMTP_Comments($SMTP_ID;"Sent via 4D"))))
   :(Not(ERRCHECK("SMTP_AddHeader";SMTP_AddHeader($SMTP_ID;"X-4Ddemo:";◊VERSION))))
   :(Not(ERRCHECK("SMTP_Body";SMTP_Body($SMTP_ID;vMessage))))
   :(Not(ERRCHECK("SMTP_Send";SMTP_Send($SMTP_ID))))
   Else
      $SentOK:=True `message was composed and mailed successfully
End case

If($SMTP_ID#0) `If a Message Envelope was created we should clear it now
   $OK:=ERRCHECK("SMTP_Clear";SMTP_Clear($SMTP_ID))
End if
```

Note: For more information about this particular use of the **Case of** structure, please refer to **Appendix A, Programming Tips**.

Below is the code for the method ***ERRCHECK***. This method takes two parameters, the name of the command ($Command), and the error value (passed by executing the command in the parameter of the method. ***ERRCHECK*** returns a boolean value corresponding to whether the error was zero. If the error is not zero, the return value ($0) gets false, otherwise it is true.

```
C_TEXT(vErrorMsg)
$Command:=$1
$Error:=$2
$Result:=True
If($Error#0)
   $Result:=False
   If(◊SHOWERRORS) `Boolean to determine whether to display error messages
      vErrorMsg:=IT_ErrorText($Error)
      ALERT("ERROR ---"+Char(13)+"Command: "+$Command+Char(13)+"Error
      Code"+String($Error)+Char(13)+"Description"+vErrorMsg)
   End if
End if
$0:=$Result
```

SMTP_Cc

SMTP_Cc (smtp_ID ; carbonCopy {; deleteOption}) -> Function result

Parameter	Type		Description
smtp_ID	Longint	⇒	Message reference
carbonCopy	Text	⇒	MailAddress or AddressList
deleteOption	Integer	⇒	0 = Add, 1 = Replace, 2 = Delete
Function result	Integer	↩	Error Code

Description

The *SMTP_Cc* command adds carbon copy recipients to the message specified by *smtp_ID*. It is not mandatory to have any addresses in the Cc: field. All addresses listed in the "To" and "cc" headers in a mail message are visible to each recipient of the message.

smtp_ID is the long integer reference to a message created with the *SMTP_New* command.

carbonCopy is a text value containing an AddressList of one or more mail addresses.

deleteOption is an integer value which specifies whether to add or delete the "Cc" header:

- A value of zero will add the new value to the "Cc" field.
- A value of 1 will set the "Cc" field to the new value, overriding any prior settings (if you pass an empty string in *carbonCopy*, the header will be removed from the mail envelope).
- A value of 2 will delete any address previously defined for the "Cc" field and remove the header from the mail envelope.
  deleteOption is an optional parameter which will default to zero if not otherwise specified.

Example

See the example for the command *SMTP_Body*.

SMTP_Charset

SMTP_Charset (encodeHeaders ; bodyCharset) -> Function result

Parameter	Type		Description
encodeHeaders	Integer	→	-1 = Use current settings, 0 = Use default settings, 1 = Convert using the specified charset
bodyCharset	Integer	→	-1 = Use current settings, 0 = Use default settings, 1= Convert using the specified charset
Function result	Integer	↩	Error Code

Description

The **SMTP_Charset** command allows automatic support of messages containing extended characters while sending them with the *SMTP_QuickSend* or *SMTP_Send* commands.

The **SMTP_Charset** command allows first, to define if the *SMTP_SetPrefs charset&Encoding* parameter value must be applied to convert the message headers, attachment filenames and body, and second, to define if a header (or attachment filename) containing extended characters must be encoded using the “=?ISO-8859-1?Q?Test=E9?= …” syntax as specified in the RFC1342. This command has an interprocess scope and will have effect on all subsequent messages sent using the *SMTP_QuickSend* and *SMTP_Send* in any 4D process.

This command is particularly useful for supporting extended characters included in the message headers such as Subject or mail addresses (for example, address encoding such as “=?ISO-8859-1?Q?Test=E9?= <test@n.net >”).

The message headers and attachment filenames will be encoded as follows, according to RFC 1342:

- Subject, Comment and attachment filenames: the full string is encoded in base 64 if it includes extended characters.
- From, To, CC, Bcc, Sender, ReplyTo, InReplyTo:
  - Any text between angle brackets (“<”, “>”) is systematically considered as an e-mail address and is not encoded.
  - Special and delimiter characters such as SPC < > ( ) @ , ; : " / ? . = are not encoded.
  - Strings delimited by special and delimiter characters are encoded in base 64 if they include extended characters.
  - Address examples:
    someone@somewhere is not encoded;
    Michèle <michele@somewhere>, only Michèle is encoded.
- Other headers are not encoded.

The *encodeHeaders* parameter specifies how to handle header conversion and encoding while sending a message. Default value is set to 0.

- -1: Use current settings;
- 0: Default value: charset is UTF-8 for subject, ISO-8859-1 for other fields;
- 1: The charset for headers and attachment filenames is defined by the **SMTP_SetPrefs** *charset&Encoding* parameter

Note: Extended headers such as “X_…” must use exclusively US ASCII codes.

The *bodyCharset* parameter specifies how to handle the message body character set and encoding conversion and encoding while sending a message. Default value is set to 0.

- -1: Use current settings;
- 0: Default value: UTF-8 base 64;
- 1: Use charset and encoding defined by the **SMTP_SetPrefs** *charset&Encoding* parameter

Note: We recommend using default settings, which are appropriate for most current systems/applications.

Example

In this example, the subject and the body are converted using the UTF-8 character set, and the subject is encoded following the RFC 1342 syntax:

```
  SMTP_SetPrefs(1;1;0)
  $err:=SMTP_Charset(1;1)
  $err:=SMTP_QuickSend("mymail.com";"myaddress";"destination";"the Euro €";"the Euro symbol is
 €")
```

SMTP_Clear

SMTP_Clear (smtp_ID) -> Function result

Parameter	Type		Description
smtp_ID	Longint	→	Message reference
		←	0 if successful
Function result	Integer	↩	Error Code

Description

The *SMTP_Clear* command disposes of a message, freeing any memory using during its creation. Every call to *SMTP_New* should have a corresponding call to *SMTP_Clear*.

smtp_ID is the long integer reference to the mail message created with the *SMTP_New* command. Upon the successful close of a SMTP message, the *SMTP_Clear* command will return a zero value back into the *smtp_ID* variable.

Example

See the example for the command *SMTP_Body*.

SMTP_Comments

SMTP_Comments (smtp_ID ; comments {; deleteOption}) -> Function result

Parameter	Type		Description
smtp_ID	Longint	→	Message reference
comments	Text	→	Comment text
deleteOption	Integer	→	0 = Replace (if comments not empty), 1 = Replace, 2 = Delete
Function result	Integer	↩	Error Code

Description

The *SMTP_Comments* command lets the user add text comments to the message while keeping the message's body untouched. The comments only appear within the header section of the message. Many mail readers do not display the full text of the message header to their users.

smtp_ID is the long integer reference to the mail message created with the *SMTP_New* command.

comments is a text value containing information you would like placed in the mail header.

Warning: The text should not contain a line feed (ascii=10). Doing so would signify the end of the header section and the beginning of the body. Subsequent header items could be pushed into the body and not recognized properly by the server or client software. For more information regarding the headers, please refer to RFC#822.

deleteOption is an integer value which specifies whether to replace or delete the "Comments" header:

- A value of zero will set the "Comments" field to the new value, overriding any prior settings (if you pass an empty string in *comments*, the prior header will be kept).
- A value of 1 will set the "Comments" field to the new value, overriding any prior settings (if you pass an empty string in *comments*, the header will be deleted).
- A value of 2 will delete any comments previously defined for the "Comments" field and remove the header from the mail envelope.
  deleteOption is an optional parameter which will default to zero if not otherwise specified.

Example

See the example for the command *SMTP_Body*.

SMTP_Date

SMTP_Date (smtp_ID ; msgDate ; msgTime ; timeZone ; offset {; deleteOption}) -> Function result

Parameter	Type		Description
smtp_ID	Longint	→	Message reference
msgDate	Date	→	Date this message was created
msgTime	Time	→	Time this message was created
timeZone	Integer	→	Location code
offset	Integer	→	Dependent on value in timeZone parameter
deleteOption	Integer	→	0 = Add/Replace, 1 = Delete
Function result	Integer	←	Error Code

Description

Given a date, a time, and a geographical location of the mail creator, the *SMTP_Date* command will build the date header for the message specified by the *smtp_ID* value. The date that is passed to the command should be the date and time for the current location of the machine sending the message. Since the parameters below must follow a specific format, the mail server on the receiving end of the message can interpret the date and time based on the date, time, time zone, and offset passed to it. It can then convert the sender's date and time to a local-time equivalent.

Note: If a mail message is composed without a Date header, the SMTP server will add one with its current date & time settings. All SMTP mail messages contain a date header, either added by the client application or the SMTP server.

smtp_ID is the long integer reference to the mail message created with the *SMTP_New* command.

msgDate is a 4D date which contains the date that this message was created.

msgTime is a time which contains the time this message was created.

timeZone identifies the time zone of the sender. This field accepts a value between zero and 6 based on the tables below.

- A value of 0 (zero) allows the user to directly specify in the *offset* parameter the number of hours to add or subtract from Universal Time.
- A value of 1 will have the sending machine automatically add the offset based on the Macintosh's PRAM. If the *timeZone* is 1 the *offset* parameter is not needed. The time zone of a Macintosh computer is determined by the settings in the **Date&Time** system preferences. Developers should give consideration to the accuracy of this option if the time values are a critical factor to their databases.
- Values 2 through 5 correspond to the 4 time zones in the United States. The *offset* for each of these values specify whether that time zone is in daylight saving time (*offset* = 1) or not (*offset* = 0).
- A value of 6 specifies that the time supplied will be military time. For this instance, the Military Time table below determines *offset*. Use the corresponding offset value (-12 through 12) based on the military time code of the location of the sender.

offset - The value of this parameter is dependent upon the code set in the *timeZone* parameter. See the descriptions above or the table below to find the correct value to set for this parameter.

Code	Time Zone	Offset Parameter
0	+/- offset from UT	Offset is in +/- Hours
1	+/- offset from UT	Offset not used, offset is supplied by Mac's PRAM
2	EST - EDT	(0 = EST, 1 = EDT)
3	CST - CDT	(0 = CST, 1 = CDT)
4	MST - MDT	(0 = MST, 1 = MDT)
5	PST - PDT	(0 = PST, 1 = PDT)
6	Military Time	See Table Below

Offset Values	Military Time Codes
0	Z
-1 thru -9	A thru I
-10 thru -12	K thru M
1 thru 12	N thru Y

Definitions of Abbreviations	
UT	Universal Time
EST	Eastern Standard Time
EDT	Eastern Daylight Time
CST	Central Standard Time
CDT	Central Daylight Time
MST	Mountain Standard Time
MDT	Mountain Daylight Time
PST	Pacific Standard Time
PDT	Pacific Daylight Time

deleteOption - A value of zero will add the date header with the given parameters, or replace a previously added set of values. A value of 1 causes any previous definition of this field to be removed. Any values in the other parameters are ignored. *deleteOption* is an optional parameter which will default to zero if not otherwise specified.

SMTP_Encrypted

SMTP_Encrypted (smtp_ID ; encrypted {; deleteOption}) -> Function result

Parameter	Type		Description
smtp_ID	Longint	⇒	Message reference
encrypted	Text	⇒	Encryption method
deleteOption	Integer	⇒	0 = Replace (if encrypted not empty), 1 = Replace, 2 = Delete
Function result	Integer	⇐	Error code

Description

The *SMTP_Encrypted* command informs the user of the type of encryption used on the body of the message. 4D Internet Commands **do not** provide the ability to encrypt or decrypt mail messages. The encryption of a message body is left to the developer. If steps are taken to encrypt the message body (prior to its assignment via *SMTP_Body*), this command should be used to identify the encryption method employed.

smtp_ID is the long integer reference to the mail message created with the *SMTP_New* command.

encrypted is a text value which specifies the type of encryption method used to encrypt the body of the message. The recipients mail software, which determines the method needed to decrypt the message body uses the encrypted header. For specific formatting requirements, please consult RFC#822.

Warning: The text should not contain a line feed (ascii=10). Doing so would signify the end of the header section and the beginning of the body. Subsequent header items could be pushed into the body and not recognized properly by the server or client software. For more information regarding the headers, please refer to RFC#822.

deleteOption is an integer value which specifies whether to replace or delete the "Encrypted" header:

- A value of zero will set the "Encrypted" field to the new value, overriding any prior settings (if you pass an empty string in *encrypted*, the prior header will be used).
- A value of 1 will set the "Encrypted" field to the new value, overriding any prior settings (if you pass an empty string in *encrypted*, the header will be deleted).
- A value of 2 will remove the "Encrypted" field from the mail envelope.
  deleteOption is an optional parameter which will default to zero if not otherwise specified.

SMTP_From

SMTP_From (smtp_ID ; msgFrom {; deleteOption}) -> Function result

Parameter	Type		Description
smtp_ID	Longint	→	Message reference
msgFrom	Text	→	MailAddress or AddressList
deleteOption	Integer	→	0 = Add, 1 = Replace, 2 = Delete
Function result	Integer	←	Error Code

Description

The *SMTP_From* command contains the mail address(es) of the person(s) to be listed in the "From" field of the message. The addresses in this field are those of the persons responsible for creating or authorizing the message. Normally, the "From" header contains one address representing the person who composed and sent the message. There may be circumstances however in which a message is composed by a group of people who should each be individually recognized within the "From" header.

The "From" header is mandatory. If an address is specified in the "From" header the existence of the "Sender" header is optional.

smtp_ID is the long integer reference to the mail message created with the *SMTP_New* command.

msgFrom is a text value containing an AddressList of one or more mail addresses. All addresses listed in the From header are visible to the recipients of the message.

Auto-Reply note: If a "ReplyTo" header is not defined for the message identified by *smtp_ID* then all replies to the message will be directed back to each person specified in the "From" header.

deleteOption is an integer value which specifies whether to add or delete the "From" header:

- A value of zero will add the new value to the "From" field.
- A value of 1 will set the "From" field to the new value, overriding any prior settings (if you pass an empty string in *msgFrom*, the header will be removed from the mail envelope).
- A value of 2 will delete any address previously defined for the "From" field and remove the header from the mail envelope.
  deleteOption is an optional parameter which will default to zero if not otherwise specified.

Example

In this example, three people compose a message on the subject of a company policy change that is distributed to everyone in the company. Any responses to this message would be directed back to each of the three people listed in the "From" header.

```
$From:="prez@acme.com, vp@acme.com, cfo@acme.com"
$Error:=SMTP_From($smtp_id;$From;0)
$Error:=SMTP_Subject($smtp_id;"Company Policy Change";0)
$Error:=SMTP_To($smtp_id;◊AllEmployee;0)
```

SMTP_GetPrefs

SMTP_GetPrefs (lineFeeds ; charset&Encoding ; lineLength) -> Function result

Parameter	Type		Description
lineFeeds	Integer	⇐	0 = Don't Add, 1 = Add LineFeeds
charset&Encoding	Longint	⇐	Charset of the message body, headers and attachment filenames as well as encoding of the body
lineLength	Longint	⇐	Maximum line length
Function result	Integer	↩	Error Code

Description

The *SMTP_GetPrefs* command returns the current settings assigned to the SMTP preferences. The values will be at their default state unless a prior call to *SMTP_SetPrefs* altered the settings. For a more complete description of the parameters, see *SMTP_SetPrefs*.

lineFeeds returns the current setting determining how the SMTP commands will handle carriage returns within the body of a message.

charset&Encoding returns the current setting for the body, headers and attachment filenames. See the *charset&Encoding* table under *SMTP_SetPrefs* for a description of combinations corresponding to values.

lineLength returns the current setting for the maximum line length of text in the message body.

SMTP_Host

SMTP_Host (smtp_ID ; hostName {; deleteOption}) -> Function result

Parameter	Type		Description
smtp_ID	Longint	⇨	Message reference
hostName	String	⇨	Host name or IP address
deleteOption	Integer	⇨	0 = Add or Replace, 1 = Delete
Function result	Integer	⮌	Error Code

Description

All mail created and sent from the SMTP commands must be directed to a specific SMTP server. 4D Internet Commands do not deliver mail directly to each recipient; it is delivered to the SMTP server specified by this command. The SMTP server is responsible for resolving address errors and scheduling the delivery of the message.

smtp_ID is the long integer reference to the mail message created with the *SMTP_New* command.

hostName is the host name or IP address of the SMTP server which will handle the distribution of the message.

deleteOption is an optional parameter which specifies whether to delete the current host setting. A value of zero will set the host to the value specified by *hostName*. A value of 1 will delete the Host specification for the message identified by *smtp_ID*. This is an optional parameter and will default to zero if not otherwise specified.

Example

See the examples for the command *SMTP_Body* and *SMTP_Send*.

SMTP_InReplyTo

SMTP_InReplyTo (smtp_ID ; inReplyTo {; deleteOption}) -> Function result

Parameter	Type		Description
smtp_ID	Longint	→	Message reference
inReplyTo	Text	→	In-Reply-To Text
deleteOption	Integer	→	0 = Replace (if inReplyTo not empty), 1 = Replace, 2 = Delete
Function result	Integer	↩	Error Code

Description

The *SMTP_InReplyTo* command identifies the previous correspondence for which this message is a response.

smtp_ID is the long integer reference to the mail message created with the *SMTP_New* command.

inReplyTo is a text value which references previous correspondences to which this message pertains. For specific formatting requirements, please consult RFC#822.

Warning: The text should not contain a line feed (ascii=10). Doing so would signify the end of the header section and the beginning of the body. Subsequent header items could be pushed into the body and not recognized properly by the server or client software. For more information regarding the headers, please refer to RFC#822.

deleteOption is an integer value which specifies whether to replace or delete the "ReplyTo" header:

- A value of zero will set the "ReplyTo" field to the new value, overriding any prior settings (if you pass an empty string in *inReplyTo*, the prior header will be kept).
- A value of 1 will set the "ReplyTo" field to the new value, overriding any prior settings (if you pass an empty string in *inReplyTo*, the header will be deleted).
- A value of 2 will delete any address previously defined for the "ReplyTo" field and remove the header from the mail envelope.
  deleteOption is an optional parameter which will default to zero if not otherwise specified.

SMTP_Keywords

SMTP_Keywords (smtp_ID ; keywords {; deleteOption}) -> Function result

Parameter	Type		Description
smtp_ID	Longint	→	Message reference
keywords	Text	→	Keywords List
deleteOption	Integer	→	0 = Replace (if keywords not empty), 1 = Replace, 2 = Delete
Function result	Integer	↩	Error Code

Description

The *SMTP_Keywords* command is used to insert keywords into the "Keywords" header of the message designated by the *smtp_ID* field.

smtp_ID is the long integer reference to the mail message created with the *SMTP_New* command.

keywords is a text value which contains a keyword or keyword list. For specific formatting requirements, please consult RFC#822.

Warning: The text should not contain a line feed (ascii=10). Doing so would signify the end of the header section and the beginning of the body. Subsequent header items could be pushed into the body and not recognized properly by the server or client software. For more information regarding the headers, please refer to RFC#822.

deleteOption is an integer value which specifies whether to replace or delete the "Keywords" header:

- A value of zero will set the "Keywords" field to the new value, overriding any prior settings (if you pass an empty string in *keywords*, the prior header will be kept).
- A value of 1 will set the "Keywords" field to the new value, overriding any prior settings (if you pass an empty string in *keywords*, the header will be deleted).
- A value of 2 will delete any keywords previously defined for the "Keywords" field and remove the header from the mail envelope.
  deleteOption is an optional parameter which will default to zero if not otherwise specified.

SMTP_MessageID

SMTP_MessageID (smtp_ID ; message_ID {; option}) -> Function result

Parameter	Type		Description
smtp_ID	Longint	⇒	Message reference
message_ID	Text	⇒	Unique ID of message
option	Integer	⇒	0 = Add (default), 1 = Replace, 2 = Delete
Function result	Integer	⇐	Error code

Description

The **SMTP_MessageID** command add a "message-id" field in the header of the message whose reference is passed in *smtp_ID*. This unique ID is used in particular on forums or public mailing lists. In general, mail servers automatically add this header to the messages they send. You can use this command to define its contents.

smtp_ID contains the ID of an e-mail created with the **SMTP_New** command.

In *message_ID*, you pass an ID to associate with the message. The contents you pass is normally unrestricted, however conventionally, it is usually in the form "*lettersOrNumbers@domainname*", for example "abcdef.123456@4d.com". Note that certain e-mail servers (such as Gmail) do not recognize custom "message-id" headers and replace them when they are not in this form.

The *option* parameter lets you specify whether any existing *message_ID* header should be kept or removed:

- If you pass 0 (default value if parameter is omitted), the contents of the parameter passed are added to the existing contents.
- If you pass 1, the contents of the parameter passed replace the existing contents.
- If you pass 2, the existing contents are removed from the message.

Example

In this example, a message with a specific "message-id" header is sent for each record of the [Admin] table:

```
$error:=SMTP_New($smtp_id)
$error:=SMTP_Host($smtp_id;"infoserv.com")
$error:=SMTP_From($smtp_id;"info@infoserv.com")
$error:=SMTP_Subject($smtp_id;"General statistics")
FIRST RECORD([Admin])
For($i;1;Records in selection([Admin]))
   $error:=SMTP_Body($smtp_id;$Stats)
   $error:=SMTP_To($smtp_id;[Admin]Email;1) // Replaces the "A" header with a new value
   $error:=SMTP_MessageID($smtp_id;[Admin]ID+"@infoserv.com";1) // Use of the admin ID
   $error:=SMTP_Send($smtp_id)
   NEXT RECORD([Admin])
End for
$error:=SMTP_Clear($smtp_id)
```

SMTP_New

SMTP_New (smtp_ID) -> Function result

Parameter	Type		Description
smtp_ID	Longint	⇐	Reference to this new message
Function result	Integer	↩	Error Code

Description

The *SMTP_New* command should be the first command called in any sequence that is going to build a SMTP mail message except where *SMTP_QuickSend* is being used. *SMTP_New* creates a new message in memory and returns a reference to the message in the *smtp_ID* long integer variable. Subsequent SMTP commands will use the *smtp_ID* reference to populate the message with header and body information prior to calling *SMTP_Send*.

Every call to *SMTP_New* should have a corresponding call to *SMTP_Clear*. After sending a message, the call to *SMTP_Clear* will free any memory held by the contents of the message.

smtp_ID is the long integer reference to the message just created. This ID will be used for all subsequent references to this message. It is possible to open multiple new messages and the *smtp_ID* returned for each provides a means of identifying which open message any subsequent command should be applied to.

Examples

See the examples for the command *SMTP_Body* and *SMTP_Send*.

SMTP_QuickSend

SMTP_QuickSend (hostName ; msgFrom ; msgTo ; subject ; message {; sessionParam}{; port}{; userName ; password}) -> Function result

Parameter	Type		Description
hostName	String	⇒	Host name or IP address
msgFrom	Text	⇒	MailAddress or AddressList
msgTo	Text	⇒	MailAddress or AddressList
subject	Text	⇒	Subject of message (UTF-8 by default)
message	Text	⇒	Message (UTF-8 by default)
sessionParam	Longint	⇒	0 or omitted = Do not use SSL but switchover allowed, 1 = Use SSL, 2 = Never use SSL (switchover not allowed), 4 = Send HTML text without SSL, 5 = Send HTML text with SSL, 8 = Send Mime HTML without SSL/TLS, 9 = Send Mime HTML with SSL/TLS
port	Longint	⇒	Number of port to use
userName	Text	⇒	User name for authentication
password	Text	⇒	Password for authentication
Function result	Integer	↩	Error Code

Description

The **SMTP_QuickSend** command gives the users the ability to build and send a mail message with one command. In the event that you require greater control over your message, or the message is of a more sophisticated nature, the group of SMTP commands based on the **SMTP_New** command should be utilized.

Note: This command cannot be used in converted databases operating in "non-Unicode" mode.

hostName is the host name or IP address of the SMTP server where the message will be sent for distribution.

msgFrom is a text value containing an AddressList of one or more complete mail addresses indicating who originally sent the message. All addresses listed in the From header are visible to the recipients of the message.

msgTo contains an AddressList value of one or more complete mail addresses. The addresses identified in the *msgTo* header will each be sent an original copy of the message. Each recipient of the message will see any other mail addresses the message was delivered to.

subject is a text value concisely describing the topic covered in detail by the message body.

Note: By default, the subject and body of the message are encoded in UTF-8, which ensures that the characters sent will be interpreted correctly by almost all of the e-mail clients. If you want to use a specific character set, refer to the **SMTP_SetPrefs** and **SMTP_Charset** commands.

message is a text value containing the body of the mail message.

The optional *sessionParam* parameter sets the message format (standard text, HTML or Mime HTML) and the activation mode of the SSL protocol for the connection:

- If you pass 0 or omit this parameter, the message will be formatted as text and sent in standard, non-secure mode. If the server proposes an update to SSL/TLS after authentication, the switchover is performed automatically (SSL/TLS operation in explicit mode).
- If you pass 1, the message will be formatted as text and sent in SSL (synchronous mode),
- If you pass 2, the message will be formatted as text and sent in standard mode but without supporting updating to SSL/TLS,
- If you pass 4, the message will be formatted as HTML and sent in standard mode,
- If you pass 5, the message will be formatted as HTML and sent in SSL/TLS mode,
- If you pass 8, the message will be formatted as Mime HTML and sent in standard mode,
- If you pass 9, the message will be formatted as Mime HTML and sent in SSL/TLS mode.
  Note: Mime HTML (.mht or .mhtml file extension) is a Web page archive format that can merge the HTML code as well as external resources such as images into a single document. It is supported by several browsers as well as MS Word, for example. Since this format is supported by 4D Write Pro areas, you will easily be able to save and send 4D Write Pro areas by mail including all their resources.

Note that these values correspond to usual combinations, however the *sessionParam* parameter is a actually a *bit field* and allows any custom combinations if you use **Bitwise Operators**:

Bit number	Format used if bit is 1
0	Use SSL or use default behavior, connect in clear, then upgrade to SSL if possible.
1	Never upgrade, stay in non-encrypted mode even if upgrade is possible. Bit is ignored if SSL (bit-0) is selected.
2	Message body is HTML, set the header accordingly.
3	MHTML message, bit-2 (HTML) is ignored. User is responsible to set everything up, except "To", "From", "Date", and "Subject"

The optional *port* parameter specifies the SMTP port number to use for connection with the server. The most frequently used values are:

- 25 = standard non-secure STMP port (default port when parameter is omitted)
- 465 = SMTPS (SSL/TLS) port
- 587 = standard (but secure) SMTP port; pass this port for connections with a MS Exchange server (explicit mode).

The optional *userName* and *password* parameters are used to authenticate the sender with the mail server. These parameters must be passed together. Note that the most secure authentication mode supported by the server will be used (as with the default mode of the **SMTP_Auth** command).

Example 1

Here is an example of use of this command:

```
$Host:="www.4d.com"
$ToAddress:="adupont@4d.fr"
$FromAddress:="jsmith@4d.com"
$Subject:="Sales Report"
$Message:="Can you send me the sales report for January 2009? Thanks."
$Error:=SMTP_QuickSend($Host;$FromAddress;$ToAddress;$Subject;$Message;1)
If($Error#0)
   ALERT("Error: SMTP_QuickSend"+Char(13)+IT_ErrorText($Error))
End If
```

Example 2

Example for using the command to send a secure message through a MS Exchange server:

```
$ServerName:="exchange.4d.com"
$MsgTo:="adupont@gmail.com"
$MsgFrom:="a.user@4d.com"
$Subject:="Test message"
$Message:="This is a test for sending a message in secure mode. Please do not reply."
$Error:=SMTP_QuickSend($ServerName;$MsgFrom;$MsgTo;$Subject;$Message;0;587;"a.user";"@!password@!")
```

Example 3

Sending a message in HTML with SSL/TLS:

```
$Host:="smtp.gmail.com"
$ToAddress:="john@4d.com"
$FromAddress:="harry@gmail.com"
$Subject:="Message HTML"
$Message:="Let’s meet at <b>Joe’s Coffee Shop</b>!"
$Param:=5 //HTML with SSL
$Port:=465 //SSL port of gmail
$User:="harry@gmail.com"
```

```
    $Password:="xyz&@!&@"
    $Error:=SMTP_QuickSend($Host;$FromAddress;$ToAddress;$Subject;$Message;$Param;$Port;$User;$Password)
```

Example 4

You saved a .mht document on your disk and want to send it by email. To do this, you can write:

```
    $Message:=Document to text("c:\\documents\\invitation.mht")
    $Host:="smtp.gmail.com"
    $ToAddress:="john@4d.com"
    $FromAddress:="harry@gmail.com"
    $Subject:="Let's party"
    $Param:=9 //MHTML with SSL
    $Port:=465 //SSL port of gmail
    $User:="harry@gmail.com"
    $Password:="xyz&@!&@"
    $Error:=SMTP_QuickSend($Host;$FromAddress;$ToAddress;$Subject;$Message;$Param;$Port;$User;$Password)
```

SMTP_References

SMTP_References (smtp_ID ; references {; deleteOption}) -> Function result

Parameter	Type		Description
smtp_ID	Longint	→	Message reference
references	Text	→	Reference Text
deleteOption	Integer	→	0 = Replace (if references not empty), 1 = Replace, 2 = Delete
Function result	Integer	↩	Error Code

Description

The *SMTP_References* command identifies additional correspondences that the message references.

smtp_ID is the long integer reference to the mail message created with the *SMTP_New* command.

references is a text value containing the reference text. For specific formatting requirements, please consult RFC#822.

Warning: The text should not contain a line feed (ascii=10). Doing so would signify the end of the header section and the beginning of the body. Subsequent header items could be pushed into the body and not recognized properly by the server or client software. For more information regarding the headers, please refer to RFC#822.

deleteOption is an integer value which specifies whether to replace or delete the "References" header:

- A value of zero will set the "References" field to the new value, overriding any prior settings (if you pass an empty string in *references*, the prior header will be kept).
- A value of 1 will set the "References" field to the new value, overriding any prior settings (if you pass an empty string in *references*, the header will be deleted).
- A value of 2 will delete any reference previously defined for the "References" field and remove the header from the mail envelope.
  deleteOption is an optional parameter which will default to zero if not otherwise specified.

SMTP_ReplyTo

SMTP_ReplyTo (smtp_ID ; replyTo {; deleteOption}) -> Function result

Parameter	Type		Description
smtp_ID	Longint	→	Message reference
replyTo	Text	→	MailAddress or AddressList
deleteOption	Integer	→	0 = Add to existing list, 1 = Replace old values with the new values, 2 = Remove the specified addresses
Function result	Integer	↩	Error Code

Description

The *SMTP_ReplyTo* command provides the user with the ability to control the direction of replies made to the message. Normally, all replies to a message come back to the people it was "From". By setting the "ReplyTo" header on outgoing mail you can affect the default routing of responses to the message.

For the database developer, *SMTP_ReplyTo* can be very powerful tool permitting them to control the behavior of replies to automated mail. Users may want replies sent to addresses other than those listed in the From or Sender addresses, such as a separate account created to track responses.

smtp_ID is the long integer reference to the mail message created with the *SMTP_New* command

replyTo is a text value containing an AddressList of one or more mail addresses. The addresses listed in this field will be used by the recipient's mail software as the default mail-account to direct their replies.

deleteOption is an integer value which specifies how to handle the address(es) listed in *replyTo*. A value of zero will add the new values to any previously assigned to this header. A value of 1 will replace any prior definitions with the new values. If *replyTo* is a null string, all prior values will be removed and the header deleted from the message. A value of 2 will delete the specified addresses from any previously assigned values. *deleteOption* is an optional parameter which will default to zero if not otherwise specified.

Example

In this example, 3 executives compose a message on the subject of a company policy change that is then distributed by the secretary to everyone in the company. Any responses to this message would be redirected to the secretary and "personnel_dept" and would not be seen by the executives.

```
$From:="prez@acme.com, vp@acme.com, cfo@acme.com"
$Error:=SMTP_From($smtp_id;$From;0)
$Error:=SMTP_Sender($smtp_id;"secretary@acme.com";0)
$Error:=SMTP_ReplyTo($smtp_id;"secretary@acme.com, personnel_dept@acme.com";0)
$Error:=SMTP_Subject($smtp_id;"Company Policy Change";0)
$Error:=SMTP_To($smtp_id;◊AllEmployee;0)
```

SMTP_Send

SMTP_Send (smtp_ID {; sessionParam}) -> Function result

Parameter	Type		Description
smtp_ID	Longint	⇒	Message reference
sessionParam	Longint	⇒	0 or omitted = Do not use SSL but switchover allowed, 1 = Use SSL, 2 = Never use SSL (switchover not allowed)
Function result	Integer	↩	Error Code

Description

The *SMTP_Send* command sends the message referenced by *smtp_ID* but does not clear the data from memory.

smtp_ID is the long integer reference to the mail message created with the *SMTP_New* command.

The optional *sessionParam* parameter sets the activation mode of the SSL protocol for the connection:

- If you pass 0 or omit this parameter, the message will be sent in standard, non-secure mode. If the server proposes an update to SSL/TLS after authentication, the switchover is performed automatically (SSL/TLS operation in explicit mode).
- If you pass 1, the message will be sent in SSL (synchronous mode),
- If you pass 2, the message will be sent in standard mode but without supporting updating to SSL/TLS.

Notes concerning the use of STARTTLS (explicit mode)

Beginning with version 13.2, 4D Internet Commands supports STARTTLS connections in explicit mode. This means that the connection is first made in standard mode and then "updated" to SSL/TLS after the authentication phase. Refer to example 2 for an illustration of this mechanism.

- The initial connection must be made on a non-SSL/TLS port that is not the default port (25). You need to call the **IT_SetPort** command before **SMTP_Send** in order to designate the port used for the initial SMTP connection. For a connection to a MS Exchange server, you must use port 587.
- The connection must be authenticated so you have to call the **SMTP_Auth** command. Only the LOGIN authentication mode is supported by 4D Internet Commands to communicate with a MS Exchange server. You can either pass this mode, or leave the default mode (in this case, the most secure mode available on the server is used):

```
$error:=SMTP_Auth($smtp_id;"user.name";"password";2) // OK for LOGIN mode
v$error:=SMTP_Auth($smtp_id;"user.name";"password") // OK for LOGIN mode set by the server
```

Example 1

In this example a message is created and the static elements are defined outside the scope of the 'for' loop. Then, for each record in the [People] table, the message is customized and sent.

```
$error:=SMTP_New($smtp_id)
$error:=SMTP_Host($smtp_id;"wkrp.com")
$error:=SMTP_From($smtp_id;"herb_tarlick@wkrp.com")
$error:=SMTP_ReplyTo($smtp_id;"bigguy@wkrp.com")
$error:=SMTP_Subject($smtp_id;"Discounts on Ad Space!")
FIRST RECORD([People])
For($i;1;Records in selection([People]))
   If([People]Sales2Date>100000)
      $Body:=◊BigDiscText
   Else
      $Body:=◊SmlDiscText
   End if
   $Body:=Replace string($BoilerPlate;"<Salutation>";[People]Firstname)
```

```
        $error:=SMTP_To($smtp_id;[People]Email;1) `Replace the "To" header with new value
        $error:=SMTP_Body($smtp_id;$Body)
        $error:=SMTP_Send($smtp_id)
        NEXT RECORD([People])
    End for
    $error:=SMTP_Clear($smtp_id)
```

Example 2

This example sends a test message using an Exchange server in STARTTLS:

```
    $error:=SMTP_New($smtp_id)
    $error:=SMTP_Host($smtp_id;"exchange.4d.com")
    $error:=SMTP_From($smtp_id;"username@4d.com")
    $error:=SMTP_ReplyTo($smtp_id;"username@4d.com")
    $error:=SMTP_Subject($smtp_id;"Message test")
    $error:=SMTP_Auth($smtp_id;"username";"!%@password") //use valid IDs
    $Body:="This is a test for messages sent through the Exchange, please do not reply"
    $error:=IT_SetPort(2;587) //standard STMP mode, port 587 for Exchange
    $error:=SMTP_To($smtp_id;"recipient@gmail.com")
    $error:=SMTP_Body($smtp_id;$Body)
    $error:=SMTP_Send($smtp_id;0) //Send in 'upgradable' mode
    ALERT(String($error));
```

SMTP_Sender

SMTP_Sender (smtp_ID ; msgSender {; deleteOption}) -> Function result

Parameter	Type		Description
smtp_ID	Longint	⇨	Message reference
msgSender	Text	⇨	MailAddress (1 only)
deleteOption	Integer	⇨	0 = Add, 1 = Replace, 2 = Delete
Function result	Integer	↩	Error Code

Description

The *SMTP_Sender* command adds the e-mail address of the person that sends the message. It is intended to be used when the sender is not the actual author of the message, or to indicate who among a group of authors actually sent the message. This field is not necessary if the contents of the "Sender" field would be redundant with the "From" field.

In cases where a computer program is the creator and sender of a mail message, the Sender header should reference the mail account of the real person responsible for administering the actions of the program and not the account managed by the computer program.

smtp_ID is the long integer reference to the mail message created with the *SMTP_New* command.

msgSender contains a single MailAddress to be listed in the Sender field of the message. Only **one** mail address may be specified for this header.

deleteOption is an integer value which specifies whether to add or delete the Sender header:

- A value of zero will add the new value to the Sender field.
- A value of 1 will set the Sender field to the new value, overriding any prior settings (if you pass an empty string in *msgSender*, the header will be removed from the mail envelope).
- A value of 2 will delete any address previously defined for the Sender field and remove the header from the mail envelope.
  deleteOption is an optional parameter which will default to zero if not otherwise specified.

Example

In this example, three executives compose a message on the subject of a company policy change which is then distributed by the secretary to everyone in the company. Any responses to this message would be directed back to each of the three people listed in the "From" header.

```
$From:="prez@acme.com, vp@acme.com, cfo@acme.com"
$Error:=SMTP_From($smtp_id;$From;0)
$Error:=SMTP_Sender($smtp_id;"secretary@acme.com";0)
$Error:=SMTP_Subject($smtp_id;"Company Policy Change";0)
$Error:=SMTP_To($smtp_id;◊AllEmployee;0)
```

SMTP_SetPrefs

SMTP_SetPrefs (lineFeed ; charset&Encoding ; lineLength) -> Function result

Parameter	Type		Description
lineFeed	Integer	⇒	1 = [default] Add, 0 = Don't Add, -1 = No Change
charset&Encoding	Longint	⇒	Charset of the message body, headers and attachment filenames as well as encoding of the body (-1 = No Change)
lineLength	Longint	⇒	Maximum line length (0 = [default] Auto-detect, -1 = No Change)
Function result	Integer	↩	Error Code

Description

The **SMTP_SetPrefs** command sets the preferences of messages to be sent using the SMTP commands. The command has a global scope and will affect all subsequent messages created with the SMTP commands. The configurable options affect the format of a mail message as it is sent to a SMTP server using the *SMTP_QuickSend* or *SMTP_Send* commands. The preference settings have an interprocess scope and effect mail creation in any 4D process.

SMTP servers recognize the end of a line to be a combined carriage return/line feed (CR/LF) character pair. This differs from most Mac applications, which view a single carriage return as the end of line/paragraph marker.

lineFeed is an integer value which specifies how to handle carriage returns within the body of a mail message. Passing a value of zero in this parameter will leave the message body text untouched, permitting the developer to control their own line feed additions. A value of 1 (default setting) will replace all carriage return/line feed pairings with carriage returns for you. A value of -1 will leave the current value of the preference unchanged. If you are unsure which option to choose, you should choose 1, the default value.

charset&Encoding specifies the charset used in the message body, headers and attachment filenames to be sent as well as the encoding to apply to the message body, according to the values in the table below. For example, “US-ASCII & 7 bit” (value 2) means that the message body charset currently in use is supposed to be US ASCII — includes only standard ASCII codes (0 through 127) which are common to Windows and Mac— and that 4D Internet Commands will encode the message body using the 7 bit encoding. Note that the *SMTP_SetPrefs* command does NOT convert the message body using the specified charset, this has to be managed by the user if necessary. If you want to force the charset conversion, refer to the *SMTP_Charset* command description.
If not changed, the default content type is 1.

Value	Body charset & encoding	Headers and attachment filenames charset (encoding always base64)
-1	No change	No change
0	Application & binary; no encoding	ISO-8859-1
1	Default: UTF-8 & base64	Default: UTF-8 for subject, ISO-8859-1 for other fields
2	US-ASCII & 7bit	ISO-8859-1
3	US-ASCII & quotable-printable	ISO-8859-1
4	US-ASCII & base64	ISO-8859-1
5	ISO-8859-1 & quotable-printable	ISO-8859-1
6	ISO-8859-1 & base64	ISO-8859-1
7	ISO-8859-1 & 8bit	ISO-8859-1
8	ISO-8859-1 & binary	ISO-8859-1
9	Reserved	Reserved
10	ISO-2022-JP (Japanese) & 7 bit	ISO-2022-JP
11	ISO-2022-KR (Korean) & 7 bit	ISO-2022-KR
12	ISO-2022-CN (Traditional & Simplified Chinese) & 7 bit	ISO-2022-CN
13	HZ-GB-2312 (Simplified Chinese) & 7 bit	HZ-GB-2312
14	Shift-JIS (Japanese) & base64	Shift-JIS
15	UTF-8 & quoted-printable	UTF-8
16	UTF-8 & base64	UTF-8

Note: We recommend using default settings, which are appropriate for most current systems/applications.

Warning: The € ("euro") character is not part of ISO-8859-1.

lineLength specifies a maximum SMTP line length for text within the message body. The SMTP commands will "line wrap" the body text by inserting a carriage return/line feed pair before the maximum line length when the text is encoded. Any number may be specified but RFC 2822 requires that line length does not exceed 998 and recommends a maximum of 78. A value of -1 will leave the current value unchanged.

The *lineLength* parameter defaults to zero. A value of zero will cause the SMTP commands to use the recommended values specified within the RFC definitions for the *charset&encoding*. If the *lineLength* parameter is set to zero, wrapping will occur based on the following table:

Body Type	Wrap at
Base64	76
Quoted-Printable	76
Other···	no wrapping

Line wrapping is strongly suggested since many systems and mail programs have problems handling messages containing unlimited line lengths. Also, keep in mind that mail often travels through a number of systems before reaching its final destination and any computer along the delivery path may reject a message if it is unable to handle the message's format.

Example

The following code sends a message with body in UTF-8 encoded in quotedprintable (headers remain in default charset):

```
  $err:=SMTP_SetPrefs(-1;15;-1)
  $err:=SMTP_Charset(0;1) //apply preferences
  $err:=SMTP_QuickSend("mymail.com";"myaddress";"destination";"the Euro €";"the Euro symbol is
€")
```

SMTP_Subject

SMTP_Subject (smtp_ID ; subject {; deleteOption}) -> Function result

Parameter	Type		Description
smtp_ID	Longint	→	Message reference
subject	Text	→	Subject of message
deleteOption	Integer	→	0 = Replace (if subject not empty), 1 = Replace, 2 = Delete
Function result	Integer	←	Error Code

Description

The **SMTP_Subject** command adds the subject of the message to the message referenced by *smtp_ID*. If a subject has already been added by a previous *SMTP_Subject* command, the new subject will override the previous subject.

smtp_ID is the long integer reference to the mail message created with the *SMTP_New* command.

subject is a text value concisely describing the topic covered in detail by the message body.

Notes:

- By default, the "Subject" header of the message is encoded in UTF-8, which ensures that the characters sent will be interpreted correctly by almost all of the e-mail clients. If you want to use a specific character set, refer to the **SMTP_SetPrefs** and **SMTP_Charset** commands.
- The text should not contain a line feed (ascii=10). Doing so would signify the end of the header section and the beginning of the body. Subsequent header items could be pushed into the body and not recognized properly by the server or client software. For more information regarding the headers, please refer to RFC#822

deleteOption is an integer value which specifies whether to replace or delete the "Subject" header:

- A value of zero will set the "Subject" field to the new value, overriding any prior settings (if you pass an empty string in *subject*, the prior header will be used).
- A value of 1 will set the "Subject" field to the new value, overriding any prior settings (if you pass an empty string in *subject*, the header will be deleted).
- A value of 2 will remove the "Subject" field from the mail envelope.
  deleteOption is an optional parameter which will default to zero if not otherwise specified.

Example

See the example for the command *SMTP_Body*.

SMTP_To

SMTP_To (smtp_ID ; msgTo {; deleteOption}) -> Function result

Parameter	Type		Description
smtp_ID	Longint	⇨	Message reference
msgTo	Text	⇨	MailAddress or AddressList
deleteOption	Integer	⇨	0 = Add, 1 = Replace, 2 = Delete
Function result	Integer	⮌	Error Code

Description

The *SMTP_To* command adds the identity of the primary recipients of the message. All addresses listed in the "To" and "cc" headers in a mail message are visible to each recipient of the message.

smtp_ID is the long integer reference to the mail message created with the *SMTP_New* command.

msgTo is a text value containing an AddressList of one or more mail addresses.

deleteOption is an integer value which specifies whether to add or delete the "To" header:

- A value of zero will add the new value to the "To" field.
- A value of 1 will set the "To" field to the new value, overriding any prior settings (if you pass an empty string in *msgTo*, the header will be removed from the mail envelope).
- A value of 2 will delete any address previously defined for the "To" field and remove the header from the mail envelope.
  deleteOption is an optional parameter which will default to zero if not otherwise specified.

Example

See the example for the command *SMTP_Body*.

IC TCP/IP

Low Level Routines, Overview

TCP/IP or Transmission Control Protocol/Internet Protocol, is the primary protocol used for sending data over the internet. The TCP commands included with 4D Internet Commands allow developers to establish TCP session and send and receive TCP packets via these sessions.

There are two ways to establish a TCP connection. The first way is to execute the *TCP_Open* command. This will open a connection with the domain specified on the specified port. *TCP_Open* allows the use of SSL (Secured Socket Layer) protocol which permits a secured connection. The other way to open a connection is to execute the *TCP_Listen* command. This command will open a connection with the specified domain on the specified port, and will listen for an incoming connection. The best way to determine if a connection has been established is to check the state of the session with the command *TCP_State* upon completion of the *TCP_Listen* command. A status code will be returned which will correspond to the current state of the session. From here you can send and/or receive TCP packets as you could with a connection established with *TCP_Open*.

In any case, any TCP connection opened must be closed subsequently using the *TCP_Close* command.

The low-level TCP/IP commands require advanced knowledge about the protocols of communication. Developers using these routines should have a complete understanding of any protocol they attempt to implement. Information about the various TCP/IP assigned port numbers, communication protocols, addressing requirements, etc. can be found in the RFCs.

Connection references in TCP commands

4D Internet commands allow the passing of POP3, IMAP or FTP connection references directly to low-level TCP commands and vice versa.
In fact, on the one hand, protocols are constantly evolving which leads to the creation of new commands; on the other, some software packages make their own interpretation of RFCs — rendering standardized implementations unusable. Using low-level TCP commands, developers can create the high-level functions they need themselves (instead of using existing functions or to fill in for a function that does not exist).
This significantly increases compatibility and development possibilities since developers can create their own high-level commands without having to rewrite all the commands needed for using a protocol.

In this example, the *IMAP_Capability* command is replaced by an equivalent function developed using TCP_IP commands.

- Here is the initial method using the *IMAP_Capability* command:

```
$ErrorNum:=IMAP_Login(vHost;vUserName;vUserPassword;vImap_ID)
If($ErrorNum=0)
   C_TEXT(vCapability)
   $ErrorNum:=IMAP_Capability(vImap_ID;vCapability)
   ... ` IMAP command using the vImap_ID parameter
End if
$ErrorNum:=IMAP_Logout(vImap_ID)
```

- This method can be replaced by:

```
$ErrorNum:=IMAP_Login(vHost;vUserName;vUserPassword;vImap_ID)
If($ErrorNum =0)
   C_TEXT(vCapability)
  ` TCP method using the value of the vImap_ID parameter:
   $ErrorNum:=My_IMAP_Capability(vImap_ID)
   ... ` IMAP commands using the vImap_ID parameter
End if
$ErrorNum:=IMAP_Logout(vImap_ID)
```

- Here is the code of the ***My_IMAP_Capability*** function:

```
C_LONGINT($1;$vErrorNum;$0)
C_TEXT($vSentText;$vReceivedText;vCapability)
C_TEXT($2)

$vImap_Id:=$1
$vCmd_Id:="A001" ` This command ID must be unique (cf. RFC 2060)
$MyvtRequestCmd:="CAPABILITY"
$vSentText;:=$vCmd_Id+""+$MyvtRequestCmd+Character(13)+Character(10)
$vReceivedText:=""
$vErrorNum:=TCP_Send($vImap_Id;$vSentText)
If($vErrorNum=0)
   $vErrorNum:=TCP_Receive($vImap_Id;$vReceivedText)
   Case of
      :($vErrorNum#0) `Reception error
         vCapability:=""
      :(Position($vCmd_Id+" OK ";$vReceivedText)#0)
 ` Command execution successful
         vCapability:=$vReceivedText
 ` In this example, we do not process the string received
      :(Position($vCmd_Id+" BAD ";$vReceivedText)#0)
 ` Failure of command execution (syntax error
 ` or unknown command)
         vCapability:=""
         $vErrorNum:=10096
   End case
End if
$0:=$vErrorNum
```

TCP_Close

TCP_Close (tcp_ID) -> Function result

Parameter	Type		Description
tcp_ID	Longint	→	Reference to an open TCP session
		←	0 = Session successfully closed
Function result	Integer	↩	Error Code

Description

The *TCP_Close* command closes the TCP session referenced by *tcp_ID*. If a TCP session is not closed, it will occupy one of the 64 references available for TCP sessions. If there are 64 sessions open which have not been closed, the user will not be able to open another session.

tcp_ID is a long integer reference to an open TCP session as established with either the *TCP_Open* or *TCP_Listen* command. This command will return the value of zero into the *tcp_ID* parameter upon successful close of the session.

TCP_Listen

TCP_Listen (ipAddress ; remotePort ; localPort ; timeout ; tcp_ID) -> Function result

Parameter	Type		Description
ipAddress	String	→	Local IP address to listen to or "" to listen to all available addresses
		←	Remote IP address used (if a variable containing a null string is passed)
remotePort	Integer	→	*** Parameter ignored ***
localPort	Integer	→	Local port number, 0 = find an unused port to use
		←	Used local port number (if 0 passed)
timeout	Integer	→	# of seconds to wait, 0 = wait forever
tcp_ID	Longint	←	Reference to this TCP session
Function result	Integer	↩	Error Code

Description

The *TCP_Listen* command opens a communication "socket" on the port specified by the *ipAddress* and *localPort* parameters. This command does not return control back to the 4D calling method until either a connection is made or the *timeout* period has elapsed. Though it may seem as though this would lock up your database until a connection was made, the command is friendly to other 4D processes that may be running. This command will slice time to other 4D processes you may already have running.

Most developers will want to issue this call from a method which has been spawned into its own 4D process (especially if you specify the *timeout* period to wait forever).

The *ipAddress* parameter contains the IP address used for the connection:

- You can pass a local address where incoming connections must be carried out.
- If a null string is passed in this parameter, the command listens to all the addresses available on the machine.
- If a variable containing a null string is passed in this parameter, it will also return the remote IP address used for the connection.

localPort contains the TCP port you wish to use for communication. If you pass a zero as this parameter, the command will find any unused port and pass that number back to this parameter.

timeout specifies the number of seconds this command will wait for an incoming connection. A zero in this parameter will cause the command to wait indefinitely for an incoming connection. Caution should be taken when passing a zero since control will never be returned to the calling 4D process if a connection is never made. Never pass zero to this parameter in a single-process database.

tcp_ID is the long integer reference to the session that was opened. This reference will be used in all subsequent TCP external calls that reference this session.

Any TCP connection opened using the *TCP_Listen* command must be closed later using the *TCP_Close* command.

Example

```
C_LONGINT(vTCPID)
C_LONGINT(vStatus)
$err:=TCP_Listen("";0;0;30;vTCPID)
$err:=TCP_State(vTCPID;vStatus)
If(vStatus=2) //socket is open and listening
   DoSomething
   $err:=TCP_Close(vTCPID)
End if
```

TCP_Open

TCP_Open (hostName ; remotePort ; tcp_ID ; sessionSettings) -> Function result

Parameter	Type		Description
hostName	String	→	Host name or IP address
remotePort	Integer	→	The remote port to connect to (0 = any)
tcp_ID	Longint	←	Reference to this TCP session
sessionSettings	Integer	→	TCP session settings 0 = Synchron (Default if omitted) 1 = Asynchron 2 = SSL In Use, Synchron 3 = SSL In Use, Asynchron
Function result	Integer	↩	Error Code

Description

The *TCP_Open* command initiates an outgoing TCP connection to a domain.

TCP_Open initiates a connection to the remote TCP referenced by *hostName*, on the port referenced by *remotePort* (if not 0). A long integer value will be returned to *tcp_ID*, which will be used by all subsequent TCP calls referring to the session. *TCP_Open* is set to time out in 30 seconds if no data is received by the session identified by the *tcp_ID* parameter. The default timeout value can be changed for all commands via *IT_SetTimeOut*.

hostName is the host name or IP address of the machine that you are opening a connection to.

remotePort indicates the TCP port on the machine indicated by *hostName* that with which you wish to establish a connection.

Note: After a call to *TCP_Open* (or *TCP_Listen*), *remotePort* may contain a negative value if the value passed to this parameter is above 32767. This will not disturb the connection. As a workaround, you can use an intermediate variable:

```
$v_ RemotePort:=v_ RemotePort
$err:=TCP_Open(v_ RemoteHostIPAdr;0;v_ SessionID)
```

tcp_ID is the long integer reference to the session that was opened. This reference will be used in all subsequent TCP external calls that reference this session.

sessionSettings is an optional parameter of the Integer type giving the user the ability to choose the TCP session settings. Note that these settings are applied to each TCP command called during the session. Default value sets to 0 (Synchron, not SSL).

SSL (Secured Socket Layer) is a protocol that allows secured TCP communications (see the 4D documentation for more information and for installation requirements).

Any TCP connection opened using the *TCP_Open* command must be closed later using the *TCP_Close* command.

Asynchron/Synchron

Asynchronous mode returns control to the 4D kernel immediately without waiting for the connection process to be finished (without waiting for the connection with the remote host to be established). Asynchronous mode is useful for people who do not want all the TCP commands to use 4D time.

Synchronous mode returns control to the 4D kernel (to other 4D processes) only when the connection process is finished (successfully or not).

- 0 = Synchron mode (Default mode, run as previous versions of 4D Internet Commands)
- 1 = Asynchron mode
- 2 = SSL In Use, Synchron. All TCP commands using the reference to this TCP session (tcp_ID) will run in Synchronous mode and use SSL protocol.
- 3 = SSL In Use, Asynchron. All TCP commands using the reference to this TCP session (tcp_ID) will run in Asynchronous mode and use SSL protocol.

Note: An error 10089 may be returned when passing 2 or 3 if an SSL connection cannot be opened (SLI library not found in the 4D Extensions folder).

Example

You want to connect to a Web site using Https; check that SLI is correctly installed and open a connection using the 443 port number:

```
   $vError:=TCP_Open(hostName;443;tcp_ID;2)
   ...
   $vError:=TCP_Close(tcp_ID) `Don't forget to close the session
```

TCP_Receive

TCP_Receive (tcp_ID ; text) -> Function result

Parameter	Type		Description
tcp_ID	Longint	⇒	Reference to an open TCP session
text	Text	⇐	Received Text
Function result	Integer	↩	Error Code

Description

Given a long integer reference to an established TCP Session, the *TCP_Receive* command receives packets of data into *text*.

tcp_ID is a long integer reference to an open TCP session as established with either the *TCP_Open* or *TCP_Listen* command.

text is the text received. When receiving data via TCP packets, you cannot count on all of your data being received by a single *TCP_Receive* call. The *TCP_Receive* command is usually called within a Repeat loop which continually checks on the status of the connection or is scanning for a known value.

Example

```
C_LONGINT($tcp_id)
C_TEXT($webpage;$buffer)
C_LONGINT(vState;$error)
$webpage:=""
vState:=0
Repeat
   $error:=TCP_Receive($tcp_id;$buffer)
   $error:=TCP_State($tcp_id;vState)
   $webpage:=$webpage+$buffer
Until((vState=0)|($error#0)) //until host closes connection or an error
```

TCP_ReceiveBLOB

TCP_ReceiveBLOB (tcp_ID ; blobToReceive) -> Function result

Parameter	Type		Description
tcp_ID	Longint	→	Reference to an open TCP session
blobToReceive	BLOB	←	BLOB to receive data
Function result	Integer	↩	Error Code

Description

Given a long integer reference to an established TCP session, the *TCP_ReceiveBLOB* command receives packets of data into *blobToReceive*.

This command performs the same action as *TCP_Receive*, except that it will receive data in a BLOB instead of a text, which allows bypassing the 32K text limitation. You can then receive binary objects.

tcp_ID is a long integer reference to an open TCP session as established with either the *TCP_Open* or *TCP_Listen* command.

blobToReceive is the BLOB which receives data. When receiving data via TCP packets, you cannot count on all of your data being received by a single *TCP_ReceiveBLOB* call. The *TCP_ReceiveBLOB* command is usually called within a **Repeat...Until** loop which continually checks on the status of the connection or is scanning for a known value.

Example

This example shows the typical structure of a method using *TCP_ReceiveBLOB*:

```
C_BLOB($Blob_Received;$Blob_All)
C_LONGINT($srcpos;$dstpos)
Repeat
   $Err:=TCP_ReceiveBLOB($TCP_ID;$Blob_Received )
   $Err:=TCP_State($TCP_ID;$State)
   $srcpos:=0
   $dstpos:=BLOB size($Blob_All)
 `Concatenating received Blobs
   COPY BLOB($Blob_Received;$Blob_All;$srcpos;$dstpos;BLOB size($Blob_Received))
Until(($State=0)|($Err#0))
```

TCP_Send

TCP_Send (tcp_ID ; sendText) -> Function result

Parameter	Type		Description
tcp_ID	Longint	→	Reference to an open TCP session
sendText	Text	→	Text to send
Function result	Integer	↩	Error Code

Description

The *TCP_Send* command sends data to the TCP session designated by *tcp_ID*.

tcp_ID is a long integer reference to an open TCP session as established with either the *TCP_Open* or *TCP_Listen* command.

sendText is a text value to be sent to the TCP session referenced by *tcp_ID*.

TCP_SendBLOB

TCP_SendBLOB (tcp_ID ; blobToSend) -> Function result

Parameter	Type		Description
tcp_ID	Longint	⇒	Reference to an open TCP session
blobToSend	BLOB	⇒	Blob to send
Function result	Integer	⮐	Error Code

Description

The *TCP_SendBLOB* command sends data to the TCP session designated by *tcp_ID*. This command performs the same action as *TCP_Send*, except that it will send a BLOB instead of a text, which allows bypassing the 32K text limitation. Using this command, you can send binary objects.

tcp_ID is a long integer reference to an open TCP session as established with either the *TCP_Open* or *TCP_Listen* command.

blobToSend is the BLOB to be sent to the TCP session referenced by *tcp_ID*.

Note regarding Platform Independence: If you exchange BLOBs between Macintosh and PC platforms, it is up to you to manage byte swapping issues if necessary.

Example

This example will send a BLOB to the TCP session:

```
C_BLOB($Blob_Send)
C_TEXT(v_Txt_Send)
TEXT TO BLOB(v_Txt_Send;$Blob_Send;Text without length;*)
$err:=TCP_SendBLOB(v_tcp_ID;$Blob_Send)
```

TCP_State

TCP_State (tcp_ID ; statusCode) -> Function result

Parameter	Type		Description
tcp_ID	Longint	➡	Reference to an open TCP
statusCode	Integer	⬅	TCP status code
Function result	Integer	↩	Error Code

Description

The *TCP_State* command returns an integer value corresponding to the state of a particular TCP connection.

tcp_ID is a long integer reference to an open TCP session as established with either the *TCP_Open* or *TCP_Listen* command.

statusCode is an integer variable returned which corresponds to one of the status codes below.

0	Connection Closed
2	Listening for an incoming connection
8	Established

Example

This example assumes that a valid TCP connection has already been established and is identified by the Longint value assigned to the $tcp_id variable. In this example, a command is sent to a Web server requesting a page of information and then a loop is entered to receive the results. Since Web servers automatically close their connections once they have performed their action, this example keeps receiving until the connection is dropped or an error occurs.

```
C_LONGINT($tcp_id)
C_LONGINT(vState;$err)
C_TEXT($command;$buffer;$response)
If(TCP_Send($tcp_id;$command)=0)
   vState:=0
   Repeat
      $err:=TCP_Receive($tcp_id;$buffer)
      $err:=TCP_State($tcp_id;vState)
      $response:=$response+$buffer
   Until((vState=0)|($err#0))
End if
```

IC UDP

UDP Commands, Overview

UDP (User Datagram Protocol) is an easy-to-implement protocol for sending data. It is faster and simpler than TCP (only 8 bytes of header as opposed to at least 20 bytes in TCP), but it does not offer the same level of reliability. It is useful for applications where data must arrive at their destination quickly. However, it does not allow verification of delivery, nor does it allow error-checking or recovery of data that was not delivered correctly.

Example

The following example illustrates how the list of 4D Servers running on a local network can be retrieved using UDP commands:

```
ARRAY TEXT(asHost;0)
ARRAY TEXT(asMachineName;0)
ARRAY TEXT(asService;0)
ARRAY TEXT(asDBName;0)
C_BLOB($Blob)

$Addr:="255.255.255.255"
$Port:=19813
$Offset:=32
SET BLOB SIZE($Blob;96;0)
TEXT TO BLOB("4D Server II";$Blob;Mac text without length;$Offset)

$Err:=UDP_New(0;$udpID)
$Err:=UDP_SendBLOBTo($udpID;$Addr;$Port;$Blob)
$Secs:=5
$Timeout:=Milliseconds+($Secs*1000)
Repeat
   DELAY PROCESS(Current process;6) `... in ticks
   SET BLOB SIZE($Blob;0;0)
   $PeerAddr:=$Addr
   $Err:=UDP_ReceiveBLOBFrom($udpID;$PeerAddr;$Port;$Blob)

   If(BLOB size($Blob)>0)
      $Offset:=0
      $Host:=BLOB to text($Blob;Mac C string;$Offset;32)
      $Offset:=32
      $Service:=BLOB to text($Blob;Mac C string;$Offset;32)
      $Offset:=64
      $DBName:=BLOB to text($Blob;Mac C string;$Offset;32)
      $Pos:=Find in array(asMachineName;$Host)
      If($Pos=-1)
         APPEND TO ARRAY(asHost;$PeerAddr)
         APPEND TO ARRAY(asMachineName;$Host)
         APPEND TO ARRAY(asService;$Service)
         APPEND TO ARRAY(asDBName;$DBName)
      End if
   End if
Until((Milliseconds>$Timeout)|($Err#0))
$Err:=UDP_Delete($udpID)
```

UDP_Delete

UDP_Delete (udp_ID) -> Function result

Parameter	Type		Description
udp_ID	Longint	→	UDP socket reference
Function result	Integer	↩	Error code

Description

The *UDP_Delete* command can be used to close a UDP socket that has been previously opened using *UDP_New* by passing its reference in *udp_ID*.

UDP_New

UDP_New (localPort ; udp_ID) -> Function result

Parameter	Type		Description
localPort	Integer	⇒	Local port used for UDP socket (0 = find any unused port to use)
udp_ID	Longint	⇐	UDP socket reference
Function result	Integer	↩	Error code

Description

The *UDP_New* command can be used to create a UDP socket by passing the local port number to be used in the *localPort* parameter. If you pass 0, the command will find any unused port. The UDP socket reference will be returned in the *udp_ID* parameter. Once this socket is no longer needed, remember to close it using *UDP_Delete* in order to free up memory.

UDP_ReceiveBLOBFrom

UDP_ReceiveBLOBFrom (udp_ID ; hostName ; remotePort ; blob) -> Function result

Parameter	Type		Description
udp_ID	Longint	→	UDP socket reference
hostName	Text variable	←	Name or IP address of server that responds
remotePort	Longint variable	←	Port of remote server that responds
blob	BLOB	←	BLOB received
Function result	Integer	↩	Error code

Description

The **UDP_ReceiveBLOBFrom** command can be used to receive a BLOB that is sent via the *udp_ID* socket.

Pass variables in the *hostName* and *remotePort* parameters. After the command is executed, these variables will contain, respectively, the name (or IP address) and the port number of the remote server from which the BLOB was received.

The BLOB received is returned in the *blob* parameter.

UDP_ReceiveFrom

UDP_ReceiveFrom (udp_ID ; hostName ; remotePort ; text) -> Function result

Parameter	Type		Description
udp_ID	Longint	⇒	UDP socket reference
hostName	Text variable	⇐	Name or IP address of server that responds
remotePort	Longint variable	⇐	Port of remote server that responds
text	Text	⇐	Text received
Function result	Integer	↩	Error code

Description

The **UDP_ReceiveFrom** command can be used to receive text that is sent via the *udp_ID* socket.

Pass variables in the *hostName* and *remotePort* parameters. After the command is executed, these variables will contain, respectively, the name (or IP address) and the port number of the remote server from which the text was received.

The text received is returned in the *text* parameter.

UDP_SendBLOBTo

UDP_SendBLOBTo (udp_ID ; hostName ; remotePort ; sendBlob) -> Function result

Parameter	Type		Description
udp_ID	Longint	→	UDP socket reference
hostName	String	→	Name or IP address of server
remotePort	Integer	→	Remote port to connect to (0=any)
sendBlob	BLOB	→	BLOB to be sent
Function result	Integer	↩	Error code

Description

The *UDP_SendBLOBTo* command can be used to send a BLOB using the *udp_ID* socket.

hostName is the name or IP address of the server where the BLOB will be sent.

remotePort is the number of the port to be connected to. If you pass 0, any available port will be used.

sendBlob is the BLOB to be sent.

UDP_SendTo

UDP_SendTo (udp_ID ; hostName ; remotePort ; sendText) -> Function result

Parameter	Type		Description
udp_ID	Longint	→	UDP socket reference
hostName	String	→	Name or IP address of server
remotePort	Integer	→	Remote port to connect to (0=any)
sendText	Text	→	Text to be sent
Function result	Integer	↩	Error code

Description

The *UDP_SendTo* command can be used to send text data using the *udp_ID* socket.

hostName is the name or IP address of the server where the text will be sent.

remotePort is the number of the port to be connected to. If you pass 0, any available port will be used.

sendText is the text to be sent.

IC Utilities

Utility Commands, Overview

The commands within this section provide various utilities which are supportive of the other sections of 4D Internet Commands. Many of these commands help the developer determine the environment in which a user's machine is operating, the versions of the software and the state and IP address of their computer.
Other commands within this section help the developer decipher error codes, encode and decode files, and effect the default timeout value for many of the commands in all sections.

IT_Decode

IT_Decode (fileName ; decodedFile ; decodeMode) -> Function result

Parameter	Type		Description
fileName	Text	→	LocalPath to an encoded file
decodedFile	Text	→	LocalPath file specification
		←	Path of decoded file
decodeMode	Integer	→	1 = BinHex 2 = Base64 (Data fork only) 3 = AppleSingle 4 = AppleDouble 5 = AppleSingle AND Base64 6 = AppleDouble AND Base64 7 = UUEncode 8 = MacBinary
Function result	Integer	↩	Error Code

Description

The *IT_Decode* command decodes a file using the *decodeMode* specified. The specified file will not be altered and a decoded copy will be created.

fileName takes a full pathname specification to a file you want to decode. If an null string is passed in this parameter the user will be prompted with a dialog to select a file.

decodedFile can be passed:

- a full LocalPath file specification providing a name and location for the decoded file.
- a full LocalPath folder specification indicating the folder that will receive the decoded file using the original file name.
- a null string (in this case, the *IT_Decode* command) will provide its own name for the document, located in the same folder as the file specified in the first parameter.
  Whether specified or not, the full path of the decoded document will be returned in this parameter.

decodeMode identifies which decoding method to apply to the file. The default value is 1 for binhex decoding. Other methods are:

Code	Scheme
1	BinHex
2	Base64 (Data fork only)
3	AppleSingle
4	AppleDouble
5	AppleSingle and Base64
6	AppleDouble and Base64
7	UUEncode
8	MacBinary

When decoding using AppleDouble (decodeModes 4 & 6), this command looks for a file named "%filename" for the resource fork.

IT_Encode

IT_Encode (fileName ; encodedFile ; encodedMode) -> Function result

Parameter	Type		Description
fileName	Text	⇒	A LocalPath to a file
encodedFile	Text	⇒	LocalPath file specification
		⇐	Path to resulting encoded file
encodedMode	Integer	⇒	1 = BinHex 2 = Base64 (Data fork only) 3 = AppleSingle 4 = AppleDouble 5 = AppleSingle AND Base64 6 = AppleDouble AND Base64 7 = UUEncode 8 = MacBinary
Function result	Integer	↩	Error Code

Description

The *IT_Encode* command encodes a file using the *encodeMode* specified. The specified file will not be altered and an encoded copy will be created. The name of the encoded file created will be the original file name plus a suffix appended to specify the encoding method. For Binhex encoding, the suffix ".hqx" will be appended. For Base64 encoding, the suffix ".b64" will be appended. For AppleSingle encoding, the suffix ".as" will be appended.

fileName takes a full pathname specification to a file you want to Encode. If an null string is passed in this parameter the user will be prompted with a dialog to select a file.

encodedFile can be passed:

- a full LocalPath file specification providing a name and location for the encoded file.
- a full LocalPath folder specification (without specifying the file name) providing the folder in which the encoded file will be saved; the file name will be the original file name with a suffix defining the encoding mode.
- a null string (in this case, the *IT_Encode* command) will provide its own name for the document, placed in the same folder as the file specified in the first parameter.
  Whether specified or not, the final pathname of the encoded document will be returned in this parameter. Due to the potential for possible naming conflicts within the specified directory, you should always rely on the returned value as the true reference to the encoded file, not the original value passed into the command.

encodeMode identifies which encoding method to apply to the file. The default value is 1 for binhex encoding. Other methods are:

Code	Scheme
1	BinHex
2	Base64 (Data fork only)
3	AppleSingle
4	AppleDouble
5	AppleSingle and Base64
6	AppleDouble and Base64
7	UUEncode
8	MacBinary

When encoding using AppleDouble (encodeModes 4 & 6), two files are created named "%filename" and "filename".

IT_ErrorText

IT_ErrorText (error) -> Function result

Parameter	Type		Description
error	Integer	→	Error code returned from other commands
Function result	String	↩	Text of the error

Description

The *IT_ErrorText* command takes an integer *error* number as its only parameter and returns the String/Text description of that error. Note that this is one of the few 4D Internet Commands that does not return an Integer as its functional value.

error is the integer number of the error.

Example

The following is an example of an ***ErrorCheck*** routine that will display an alert message explaining the cause of an error.

```
 `Method: ERRCHECK ("Command Name"; Error# ) -> True/False
C_TEXT(vErrorMsg)
$Command:=$1
$Error:=$2
$Result:=True
If($Error#0)
   $Result:=False
   vErrorMsg:=IT_ErrorText($Error)
   ALERT("ERROR -- "+Char(13)+"Command: "+$Command+Char(13)+"Error Code:"+String($Error)
   +Char(13)+"Description: "+vErrorMsg)
End if
$0:=$Result
```

IT_GetPort

IT_GetPort (protocol ; port) -> Function result

Parameter	Type		Description
protocol	Integer	⇒	1 = FTP; 2 = SMTP; 3 = POP3; 4 = IMAP; 12 = SMTP SSL ; 13 = POP3 SSL ; 14 = IMAP SSL
port	Integer	⇐	Port Number
Function result	Integer	↩	Error Code

Description

Given a specified *protocol*, the *IT_GetPort* command will get the current *port* number being used by the 4D Internet Commands related to the protocol.

IT_GetProxy

IT_GetProxy (protocol ; proxyKind ; proxyHostName ; proxyPort ; proxyUserID) -> Function result

Parameter	Type		Description
protocol	Integer	→	1 = FTP; 2 = SMTP; 3 = POP3; 4 = IMAP
proxyKind	Integer	←	0 = None; 1 = SOCKS
proxyHostName	String	←	Host name or IP address of the SOCKS Proxy host
proxyPort	Integer	←	Proxy port to connect to
proxyUserID	Text	←	UserID for SOCKS
Function result	Integer	↩	Error Code

Description

Given a specified protocol, the *IT_GetProxy* command returns the current settings being used by the 4D Internet Commands related to the routage of the specified protocol. The values will be at their default state unless a prior call to *IT_SetProxy* altered the settings. For a complete description of the parameters, see *IT_SetProxy*.

protocol is an integer value that specifies the protocol to examine. A value of 1 will indicate FTP protocol. A value of 2 will indicate SMTP protocol. A value of 3 will indicate POP3 protocol. A value of 4 will indicate IMAP protocol.

proxyKind returns the current settings determining if a SOCKS proxy host is used. A value of 1 routes all requests for the specified protocol through the specified SOCKS Host. A value of zero does not route requests for the specified protocol through any SOCKS Host.

proxyHostName returns the current settings determining the Host Name or IP address of the SOCKS Proxy host in use.

proxyPort returns the current settings determining the port number used for the specified protocol to communicate with the SOCKS Proxy host.

proxyUserID returns the current settings determining the user ID.

IT_GetTimeOut

IT_GetTimeOut (timeout) -> Function result

Parameter	Type		Description
timeout	Integer	←	Timeout seconds
Function result	Integer	↩	Error Code

Description

The *IT_GetTimeOut* command returns the current timeout value for the commands listed in *IT_SetTimeOut*.
timeout is the current value in seconds of the timeout period.

IT_MyTCPAddr

IT_MyTCPAddr (ip_Address ; subnet) -> Function result

Parameter	Type		Description
ip_Address	String	⇐	IP address of users machine
subnet	String	⇐	Subnet Mask in IP form
Function result	Integer	⮌	Error Code

Description

The *IT_MyTCPAddr* command returns the IP address of the machine that executes the command.

ip_Address is the string returned which contains the IP address.

subnet is the string returned which contains the Subnet mask of the IP address.

IT_Platform

IT_Platform -> Function result

Parameter	Type		Description
Function result	Integer	↩	Platform Type (1 = Mac OS, 2 = Windows)

Description

The *IT_Platform* function returns an integer value indicating which set of 4D Internet Commands code is currently executing. The function will return a one if running on Mac OS or a 2 if running on Windows.

Example

```
C_BOOLEAN(◊ITnative)
◊ITnative:=(IT_Platform=1)
```

IT_PPPConnect

IT_PPPConnect (pppProfil) -> Function result

Parameter	Type		Description
pppProfil	String	→	Dial-up name = Null string on Mac OS, fill under Windows
Function result	Integer	↩	Error code

Description

The *IT_PPPConnect* command opens the current dial-up connection under Mac OS or the specified dial-up connection (*pppProfil*) parameter under Windows. This command acts as a function and returns an integer value error if the connection cannot be opened.

This command must be executed each time you need to execute a set of Internet Commands that works online. On completion, you must execute *IT_PPPDisconnect* to close the current connection.

PPP (Point-to-Point Protocol) is a protocol for communication between two computers using a serial interface, typically a personal computer connected by a phone line to a server. For instance, your Internet server provider may supply you with a PPP connection so that the provider’s server can respond to your requests, pass them on to the Internet and forward your requested Internet responses back to you. Essentially, it packages your computer’s TCP/IP packets and forwards them to the server where they can actually be served on the Internet.

PPP is usually preferred over the former de facto standard Serial Line Internet Protocol (SLIP) because it can handle synchronous as well as asynchronous communication. PPP can share a line with other users and has an error detection function, neither of which is true for SLIP. If a choice is possible, PPP is preferred.

IT_PPPDisconnect

IT_PPPDisconnect (pppProfil) -> Function result

Parameter	Type		Description
pppProfil	String	→	Dial-up name = Null string on Mac OS, optionally filled under Windows
Function result	Integer	↩	Error code

Description

The *IT_PPPDisconnect* command closes the current dial-up connection previously opened by *IT_PPPConnect*.
pppProfil is a text value specifying the dial-up connection to close.
Under Windows, this parameter may be useful when several PPP connections are opened simultaneously. Using this parameter will ensure good running whatever the user network configuration.

Under Windows

- If only one connection is opened and *pppProfil* is not passed or is passed as a null string, *IT_PPPDisconnect* closes the opened connection.
- If several connections are opened and *pppProfil* is not passed or is passed as a null string, *IT_PPPDisconnect* returns an error and does not close any connection.
- If *pppProfil* is passed and valid, the specified connection is closed whatever the number of opened connections.

Under Mac OS

This parameter is not taken into account.

IT_PPPStatus

IT_PPPStatus (pppProfil) -> Function result

Parameter	Type		Description
pppProfil	String	→	Dial-up name = Null string on Mac OS, optionally filled under Windows
Function result	Integer	↩	1 if connected; 0 if connecting; -1 if error

Description

The *IT_PPPStatus* command allows you to check the status of a connection opened with the *IT_PPPConnect* command or opened manually.

pppProfil is a text value specifying which opened connection to check.
Under Windows, this parameter is optional but may be useful to ensure good running whatever the user network configuration.

Under Windows

- If *pppProfil* is passed and valid, specified connection status is return.
- If *pppProfil* is not passed or is passed as a null string, *IT_PPPStatus* will return:
  - -1 if several connections are opened,
  - the status of the opened connection if only one connection is opened

Under Mac OS

This parameter is not taken into account.

IT_PPPStatus returns an integer denoting the connection status. It returns:

- 1 if connected,
- 0 if connecting,
- -1 in case of a connection failure or if not connected.

Example

```
 //Method GetMessages (this method is executed in a process)
If(mPPPConnect($vPPPProfil;120))
   $vErrCode:=IT_MacTCPInit
   If($vErrCode=0)
      $vErrCode:=POP3_Login...
      ...
   Else
      ALERT("Connection failed")
   End if
End if

 //Method mPPPConnect
C_BOOLEAN($0) //returns True if we are currently connected, False if connection failed
C_TEXT($1) //null string if Mac OS, Entry Name if Windows
C_LONGINT($2) //timeout in seconds

If(IT_PPPStatus=1)
   $0:=True //we are already connected
Else
   $vTimeoutLength:=$2
   $vTimeout:=False
   $vErr:=IT_PPPConnect($1)
   If($vErr=0)
```

```
      $vStart:=Current time
      Repeat
         DELAY PROCESS(Current process;30)
         $vStatus:=IT_PPPStatus($1)
         $vTimeout:=((Current time-$vStart)>$vTimeoutLength)
      Until(($vStatus=1)|$vTimeout) //we are connected or time out
      If(Not($vTimeout))
         $0:=True //we are connected
      End if
   End if //… $Err = 0
End if
```

IT_SetPort

IT_SetPort (protocol ; port) -> Function result

Parameter	Type		Description
protocol	Integer	⇨	1 = FTP ; 2 = SMTP ; 3 = POP3 ; 4 = IMAP ; 12 = SMTP SSL ; 13 = POP3 SSL ; 14 = IMAP SSL
port	Integer	⇨	Port Number
Function result	Integer	⮌	Error Code

Description

Given a specified *protocol*, the *IT_SetPort* command will direct all future communication of the protocol to the specified *port*.

IT_SetProxy

IT_SetProxy (protocol ; proxyKind ; proxyHostName ; proxyPort ; proxyUserID) -> Function result

Parameter	Type		Description
protocol	Integer	→	1 = FTP; 2 = SMTP; 3 = POP3; 4 = IMAP
proxyKind	Integer	→	0 = None; 1 = SOCKS
proxyHostName	String	→	Host name or IP address of the SOCKS Proxy
proxyPort	Integer	→	Proxy port to connect to
proxyUserID	Text	→	UserID for SOCKS
Function result	Integer	↩	Error Code

Description

The *IT_SetProxy* command allows you to negociate a connection using the specified protocol and then send all further requests through the SOCKs Host (SOCKS Proxy). If you are just connecting to an intranet, then you will probably not have to communicate through the SOCKS Host. However, it all depends on how your company has their firewall set up. The *IT_SetProxy* settings have an interprocess scope and effect all the connections using the specified protocol in any 4D process.

Note: Socks (or "SOCKS") is a protocol that a proxy server can use to accept requests from client users in a company's network so that it can forward them across the Internet. If your workstation is located behind a firewall and you want to access an information located on the Internet, the SOCKS host receives your request, forwards the request through the firewall, and then returns the information to your client application.

protocol is an integer value that specifies the protocol to be routed through the specified SOCKS Proxy host. A value of 1 will effect FTP protocol. A value of 2 will effect SMTP protocol. A value of 3 will effect POP3 protocol. A value of 4 will indicate IMAP protocol.

proxyKind is an integer value indicating whether the specified protocol should be routed through a SOCKS Proxy host or not. A value of 1 will route all requests for the specified protocol through the specified SOCKS Host. A value of zero won't route requests for the specified protocol through any SOCKS Host.

proxyHostName is the Host name or the IP address of the SOCKS Proxy machine.

proxyPort is an integer value that specifies the port to use for the specified protocol to communicate with the SOCKS Proxy host.

proxyUserID is a text value that identifies the user. The user ID is given by your network administrator. *proxyUserID* can be an empty text ("").

Example

Using the following method, all FTP connections will be routed through the specified SOCKS Proxy Host.

```
$err:=IT_SetProxy(1;1;$proxyAdd;$proxyPort;"") `FTP SOCKS Proxy
$err:=FTP_Login("ftp.4d.com";"anonymous";dbody@aol.com";$ftpID)
$err:=FTP_GetFileInfo($ftpID;$vpath;$vsize;$vmodDate)
$err:=FTP_Receive($ftpID;$vpath;"";0)
$err:=FTP_Logout($ftpID)
```

Note: For clarification purposes, this example does not contain error checking.

The following statement stops routing FTP connections through any SOCKS Proxy Host.

```
$err:=IT_SetProxy(1;0;$proxyAdd;$proxyPort;"")
```

IT_SetTimeOut

IT_SetTimeOut (timeout) -> Function result

Parameter	Type		Description
timeout	Integer	→	Timeout in seconds; limited to 0 thru 127
Function result	Integer	↩	Error Code

Description

The *IT_SetTimeOut* command sets the integer value of the timeout period in seconds. This value is limited to between zero and 127 seconds. By default, the timeout period is 30 seconds.

timeout is the current value in seconds of the timeout period. The following commands are affected by *IT_SetTimeOut*:

TCP_Open
FTP_Login
FTP_Send
FTP_Receive
SMTP_QuickSend
SMTP_Send
POP3_Login
POP3_BoxInfo
POP3_Delete
POP3_Reset
POP3_MsgInfo
POP3_MsgLstInfo
POP3_GetMessage
POP3_MsgLst
POP3_Download
POP3_VerifyID
POP3_UIDToNum
IMAP_Login
IMAP_VerifyID
IMAP_Capability
IMAP_ListMBs
IMAP_SubscribeMB
IMAP_GetMBStatus
IMAP_SetCurrentMB
IMAP_Delete
IMAP_MsgInfo
IMAP_MsgLstInfo
IMAP_GetMessage
IMAP_MsgLst
IMAP_SetFlags
IMAP_GetFlags
IMAP_Search
IMAP_MsgFetch
IMAP_Download
IMAP_CopyToMB
IMAP_CreateMB
IMAP_RenameMB
IMAP_DeleteMB
NET_Finger
NET_Ping

NET_Time

Note: Setting the *timeout* to zero for the *TCP_Listen* command allows it to listen indefinitely. Make sure you set the timeout back to some other value after this command. Also, the timeout value is used for "TCP/IP timeouts" AND "wait for a response timeout". If you set the timeout to zero, it will never get enough time to wait for a response.

IT_TCPversion

IT_TCPversion (stackKind ; stackVersion) -> Function result

Parameter	Type		Description
stackKind	Integer	⇐	0 = None, 3 = WinSock, 4 = BSD
stackVersion	Text	⇐	Version number of the TCP stack
Function result	Integer	↩	Error Code

Description

The *IT_TCPversion* command returns information about the type of TCP stack currently in use by the 4D Internet Commands. The type of Stack varies by platform. Under Macintosh, only BSD is now supported. Under Windows, the WinSock TCP stack is supported.

stackKind returns an integer value expressing the type of TCP stack currently in use. The value returned identifies the following supported TCP stacks:

Code	TCP Stack
0	None
1	MacTCP (Obsolete, see Compatibility note)
2	Open Transport (Obsolete, see Compatibility note)
3	WinSock
4	BSD Sockets

Compatibility notes:

- MacTCP is no longer supported (starting from version 6.8). Therefore, the *stackKind* parameter will no longer return "1".
- Open Transport is no longer supported (starting from version 2004). Therefore, the *stackKind* parameter will no longer return "2".

stackVersion returns a Text value representing the version number of the TCP stack currently in use and identified by the *stackKind* parameter.

IT_Version

IT_Version -> Function result

Parameter	Type		Description
Function result	String	⮐	Version String

Description

The *IT_Version* function returns a String value indicating the version number of 4D Internet Commands.

Example

The following example presents an alert dialog to the user indicating what version of 4D Internet Commands they are using.

```
ALERT("4D Internet Commands version: "+IT_Version)
```

IT_MacTCPInit

IT_MacTCPInit -> Function result

Parameter	Type		Description
Function result	Integer	↩	Error Code

Compatibility note

The *IT_MacTCPInit* command now has no effect and must no longer be used.

IT_MacTCPVer

IT_MacTCPVer (versionCode) -> Function result

Parameter	Type		Description
versionCode	Integer	⇐	Version Code of MacTCP installed
Function result	Integer	↩	Error Code

Compatibility note:

The command *IT_MacTCPVer* has been made obsolete with the addition of the *IT_TCPversion* which provides better capability for determining version information across Open Transport and Winsock.

Description

Starting with 4D Internet Commands version 6.8, MacTCP is no longer supported. Consequently, the *versionCode* parameter systematically returns 0 whatever the platform and/or OS used.

Appendixes

Executing Commands via a Case Statement

In many of the examples in this document, a programming construct is used which is likely to be unfamiliar to many developers. Many of these examples execute a series of commands as falsified cases within a 4D Case statement.

Many of the commands within 4D Internet Commands require that an entire sequence of commands execute successfully in order to complete. Since the failure of any one command within the sequence should stop any further processing along that path, it would become laborious to cascade your If conditions many level deeps:

```
If(SMTP_New($smtp_id)=0)
   If(SMTP_Host($smtp_id;◊pref_Server)=0)
      If(SMTP_From($smtp_id;vFrom)=0)
         If(SMTP_To($smtp_id;vTo)=0)
            and deeper, and deeper
         End if
      End if
   End if
End if
```

An alternative to this method is to rely on the manner in which 4D executes it's case statements. Each item of a case statement is executed by 4D in order to determine if its return value is **True** or **False**. If all elements of a case statement were to return a false value, then each element of that case statement's tests would have been run. We can execute the same code described above by the following:

```
$SentOK:=False `A flag to indicate if we made it through all of the calls
Case of
   :(SMTP_New($smtp_id)#0)
   :(SMTP_Host($smtp_id;◊pref_Server)#0)
   :(SMTP_From($smtp_id;vFrom)#0)
   :(SMTP_To($smtp_id;vTo)#0)
   :(SMTP_Subject($smtp_id;vSubject)#0)
   :(SMTP_Body($smtp_id;vMessage)#0)
   :(SMTP_Send($smtp_id)#0)
   Else
      $SentOK:=True `message was composed and mailed successfully
End case
If($smtp_id#0) `If a Message Envelope was created we should clear it now
   $OK:=SMTP_Clear($smtp_id)
End if
```

In the above example, every 4D Internet command will return a zero error number if it successfully completed. In order for 4D to evaluate each case statement, it must actually execute each external call to obtain its return value. Since each case element compares the return result to not zero, 4D will not find an element to stop on until one of the commands fails. If every command executes successfully, 4D will proceed down to run the **Else** condition where we set the $SentOK flag to indicate that the message was composed and sent successfully.

Suggestions when auto-replying to POP3 or IMAP mail

If you are planning on implementing a mail system within your database in which the user can "Reply" to mail they have received, there are some standard suggestions for how to fill out the fields of the reply message. The following suggestions are outlined by RFC#822:

- The address listed in the "Sender" field should receive notices of any problems during delivery of the initial messages. If no "Sender" field exists, notices should be sent to the address listed in the "From" field. The "Sender" mail address should only be sent replies pertaining to problems in the mail delivery and **not** replies related to the topic of the message.

- The "Sender" address should never be used in an automated process of replying to messages. Instead, the message should either use the "Reply-To" field or the "From" field, dependent on the conditions described below.
- If the "Reply-To" field exists and contains one or more mail addresses, then any reply should be directed to the people in that list. Addresses within the "Reply-To" header override any addresses listed in the "From" header. However, if no "Reply-To" field exists but a "From" field does exist, replies should be sent to the mailbox(es) indicated in the "From" header.

These suggestions are only meant to help the decision process when the mail addressing is programmatically handled in the case of "Reply" type actions. Once the Reply message has been created, the end-user can certainly override any of these defaults before sending the message.

Appendix B, TCP Port Numbers

How to choose a port number

- 0 to 1023 (Well Known Ports): The Well Known Ports are assigned by the I.A.N.A. (Internet Assigned Numbers Authority) and on most systems can only be used by system (or root) processes or by programs executed by privileged users.
  - 20 and 21 FTP;
  - 23 TELNET;
  - 25 SMTP;
  - 37 NTP;
  - 80 and 8080 HTTP;
  - 443 HTTPS.
- 1024 to 49151 (Registered Ports): The Registered Ports are listed by the I.A.N.A. and on most systems can be used by ordinary user processes or programs executed by ordinary users (routers, specific applications...)
- 49152 to 65535 (Dynamic and/or Private Ports) : The Dynamic and/or Private Ports are free of use.

People who want to use TCP/IP commands to synchronize databases would have to use port numbers higher than 49151.

For more information, please visit the I.A.N.A. Web site: *http://www.iana.org*

TCP Port Numbers

daytime	13	Daytime
qotd	17	Quote of the Day
ftp-data	20	File Transfer [Default Data]
ftp	21	File Transfer [Control]
telnet	23	Telnet
smtp	25	Simple Mail Transfer
time	37	Time
nicname	43	Who Is
domain	53	Domain Name Server
sql*net	66	Oracle SQL*NET
gopher	70	Gopher
finger	79	Finger
http	80	World Wide Web HTTP
poppassd	106	Password Server
rtelnet	107	Remote Telnet Service
pop2	109	Post Office Protocol - Version 2
pop3	110	Post Office Protocol - Version 3
sunrpc	111	SUN Remote Procedure Call
auth	113	Authentication Service
sftp	115	Simple File Transfer Protocol
sqlserv	118	SQL Services
nntp	119	Network News Transfer Protocol
ntp	123	Network Time Protocol
pwdgen	129	Password Generator Protocol
imap2	143	Interactive Mail Access Protocol v2
news	144	NewS
sql-net	150	SQL-NET
multiplex	171	Network Innovations Multiplex
cl/1	172	Network Innovations CL/1
at-rtmp	201	AppleTalk Routing Maintenance
at-nbp	202	AppleTalk Name Binding
at-3	203	AppleTalk Unused
at-echo	204	AppleTalk Echo
at-5	205	AppleTalk Unused
at-zis	206	AppleTalk Zone Information
at-7	207	AppleTalk Unused
at-8	208	AppleTalk Unused
ipx	213	IPX
netware-ip	396	Novell Netware over IP
timbuktu	407	Timbuktu
https	443	Secured protocol
conference	531	chat
netnews	532	readnews
netwall	533	for emergency broadcasts
uucp	540	uucpd
uucp-rlogin	541	uucp-rlogin
whoami	565	whoami
ipcserver	600	Sun IPC server
phonebook	767	phone

accessbuilder　888　AccessBuilder

Appendix C, 4D Internet Commands Error Codes

All 4D Internet Commands (with the exception of *IT_ErrorText* & *IT_Version*) return an integer value as the result of the function. This integer contains any error number which the command needs to convey back to the 4D database. If a command is successful, a zero will be returned. The source of an error number can usually be determined by the range of values which the error falls within. The following table provides an index to the most likely creator of an error in any given range:

Error Number	**Generated by**
Error < Zero	Operating System Error or WinSock network layer
Zero	No Error
Error 1 -> 61	BSD network layer
Error >= 10000	4D Internet Commands Error

4D Internet Commands Error Codes

If an error occurs during any operation, a numeric value from the following table will be returned:

10000	user cancelled a dialog or progress.
10001	unimplemented Internet command.
10002	invalid array type.
10003	no more (TCP,SMTP,POP3, etc.) references available.
10004	invalid reference.
10005	need a "Host" for use in the "SMTP_Send" command.
10006	need a "From" for use in the "SMTP_Send" command.
10007	need a recipient for use in the "SMTP_Send" command.
10008	already logged in.
10009	error trying to make a POP3 connection.
10010	error with POP3 USER.
10011	error with POP3 PASS.
10012	error with POP3 QUIT.
10013	error with POP3 STAT.
10014	error with POP3 LIST.
10015	error with POP3 UIDL.
10016	error with POP3 DELE.
10017	error with POP3 RSET.
10018	invalid message number.
10019	invalid character offset.
10020	invalid character length.
10021	error with POP3 RETR.
10022	field was not found in mail Header.
10023	no attachments found.
10024	error in processing BinHex.
10025	BinHex checksum error.
10026	Internet commands unavailable. Probably because MacTCP is not installed
10027	Connection no longer exists
10028	Exceeded 32k limit
10029	Error with POP3 NOOP
10030	POP3 session was closed by the server
10031	Error with POP3 APOP
10032	Unknown or invalid response.
10033	SMTP 421 - Service not available, closing transmission channel.
10034	SMTP 450 - Requested mail action not taken: mailbox unavailable.
10035	SMTP 451 - Requested action aborted: local error in processing.
10036	SMTP 452 - Requested action not taken: insufficient system storage.
10037	SMTP 500 - Syntax error, command unrecognized.
10038	SMTP 501 - Syntax error in parameters or arguments.
10039	SMTP 502 - Command not implemented.
10040	SMTP 503 - Bad sequence of commands.
10041	SMTP 504 - Command parameter not implemented.
10042	SMTP 550 - Requested action not taken: mailbox unavailable.
10043	SMTP 551 - User not local; please try <forward-path>.
10044	SMTP 552 - Requested mail action aborted: exceeded storage allocation.
10045	SMTP 553 - Requested action not taken: mailbox name not allowed.
10046	SMTP 554 - Transaction failed.
10047	FTP 421 - Service not available, closing control connection.
10048	FTP 425 - Can't open data connection.

10049	FTP 426 - Connection closed; transfer aborted.
10050	FTP 450 - Requested file action not taken. File unavailable (e.g.,file busy).
10051	FTP 451 - Requested action aborted: local error in processing.
10052	FTP 452 - Requested action not taken. Insufficient storage space in system.
10053	FTP 500 - Syntax error, command unrecognized.
10054	FTP 501 - Syntax error in parameters or arguments.
10055	FTP 502 - Command not implemented.
10056	FTP 503 - Bad sequence of commands.
10057	FTP 504 - Command not implemented for that parameter.
10058	FTP 530 - Not logged in.
10059	FTP 532 - Need account for storing files.
10060	FTP 550 - Requested action not taken. File unavailable (e.g., file not found, no access).
10061	FTP 551 - Requested action aborted: page type unknown.
10062	FTP 552 - Requested file action aborted. Exceeded storage allocation (for current directory or dataset).
10063	FTP 553 - Requested action not taken. File name not allowed.
10064	No response has been received within the given timeout period.
10065	Not an FTP file.
10066	Error in processing Base64.
10067	Error in processing AppleSingle.
10068	Error in processing Quoted-Printable.
10069	FTP session was closed by the server.
10070	Not an FTP directory.
10071	TCP session was closed by the server
10072	Invalid encode kind
10073	Invalid decode kind
10074	An asynchronous DNR call did not complete
10075	An asynchronous OpenTransport call did not complete
10076	OpenTransport bind failed
10077	OpenTransport connect failed
10078	Maximum MacTCP streams reached
10079	Error in processing uuencode
10080	Cannot load ICMP library
10081	Error in processing MacBinary
10082	MacBinary checksum error
10083	Could not open a file
10084	No FTP information received
10085	Unknown FTP information received
10086	Proxy connection failed
10087	Standard file I/O error
10088	FTP reentrant error
10089	SLI.DLL is not loaded
10091	Error trying to make an IMAP connection
10092	A maibox is not selected
10093	Invalid message part
10094	Error with IMAP LOGIN
10095	Error with IMAP LOGOUT
10096	Error with IMAP CAPABILITY
10097	Error with IMAP SELECT
10098	Error with IMAP FETCH

10099	Error with IMAP PARTIAL
10100	Error with IMAP STORE
10101	Error with IMAP EXPUNGE
10102	Error with IMAP SEARCH
10103	Error with IMAP COPY
10104	Error with IMAP CREATE
10105	Error with IMAP DELETE
10106	Error with IMAP RENAME
10107	Error with IMAP SUBSCRIBE
10108	Error with IMAP UNSUBSCRIBE
10109	Error with IMAP LIST
10110	Error with IMAP LSUB
10111	Error with IMAP STATUS
10112	Error with IMAP CLOSE
10113	Error with AUTHENTICATION

BSD Error Codes

1	Operation not permitted
4	Interrupted system call
13	Permission denied
14	Bad address
22	Invalid argument
24	Too many open files
35	Operation would block
36	Operation now in progress
37	Operation already in progress
38	Socket operation on non-socket
39	Destination address required
40	Message too long
41	Protocol wrong type for socket
42	Protocol not available
43	Protocol not supported
44	Socket type not supported
45	Operation not supported
46	Protocol family not supported
47	Address family not supported by protocol family
48	Address already in use
49	Can't assign requested address
50	Network is down
51	Network is unreachable
52	Network dropped connection on reset
53	Software caused connection abort
54	Connection reset by peer
55	No buffer space available
56	Socket is already connected
57	Socket is not connected
58	Can't send after socket shutdown
60	Operation timed out
61	Connection refused

WinSock Error Codes

-10004	Blocking call cancelled
-10013	Permission denied
-10014	Bad address
-10022	Invalid argument
-10024	No more sockets available
-10035	Non-blocking socket would block
-10036	Illegal WinSock function invoked while a blocking function is in progress
-10037	An attempt was made to cancel an asynchronous operation that has already completed
-10038	Specified socket descriptor is not valid for this application
-10039	Destination address was required but none was supplied to the function
-10040	Datagram too large for buffer
-10041	Specified protocol does not match the other parameters in the call
-10042	Protocol option is unknown or invalid
-10043	Specified protocol is not supported by the Windows Sockets implementation
-10044	Specified socket type is not supported by the specified address family
-10045	Socket does not support the specified operation
-10046	Protocol family not supported
-10047	Specified address family is not supported by the Windows Sockets implementation or cannot be used with the indicated socket
-10048	Specified address is already in use
-10049	Specified address is not available from the local machine
-10050	Problem with the network subsystem
-10051	Network cannot be reached from this host at this time
-10052	Connection was dropped and must be reset
-10053	Connection was aborted because of a timeout or other error condition
-10054	Connection was reset by the remote host
-10055	Windows Sockets implementation is out of buffer space or the space provided in an API call by the application was too small to hold the requested information
-	Specified socket is already connected

10056	Specified socket is already connected
-10057	Specified socket is not connected
-10058	Socket has had the requested functionality shut down
-10060	Connection attempt timed out before the connection could be established
-10061	Connection attempt was forcefully rejected
-10091	Network subsystem is not yet ready for communication
-10092	Windows Sockets DLL does not support the requested Winsock protocol version
-10093	Windows Sockets not initialized
-11001	Requested database information does not exist; as confirmed by an authoritative host
-11002	Requested information was not found but the answer was not authoritative
-11003	Non-recoverable error occurred
-11004	Name supplied was valid but no information of the requested type is in the database

SMTP RFC Values

The following items are **not** error codes returned by any of the external commands. These are response codes which the SMTP protocol has defined to communicate various states during client-server communication. Developers may find this list useful if they are writing their own mail communication procedures using low-level TCP commands.

211	System status, or system help reply
214	Help message [Information on how to use the receiver or the meaning of a particular non-standard command; this reply is useful only to the human user]
220	<domain> Service ready
221	<domain> Service closing transmission channel
250	Requested mail action okay, completed
251	User not local; will forward to <forward-path>
354	Start mail input; end with <CRLF>.<CRLF>
421	<domain> Service not available, closing transmission channel [This may be a reply to any command if the service knows it must shut down]
450	Requested mail action not taken: mailbox unavailable [e.g., mailbox busy]
451	Requested action aborted: local error in processing
452	Requested action not taken: insufficient system storage
500	Syntax error, command unrecognized [This may include errors such as command line too long]
501	Syntax error in parameters or arguments
502	Command not implemented
503	Bad sequence of commands
504	Command parameter not implemented
550	Requested action not taken: mailbox unavailable [e.g., mailbox not found, no access]
551	User not local; please try <forward-path>
552	Requested mail action aborted: exceeded storage allocation
553	Requested action not taken: mailbox name not allowed [e.g., mailbox syntax incorrect]
554	Transaction failed

FTP RFC Values

The following items are **not** error codes returned by any of the external commands. These are response codes which the FTP protocol has defined to communicate various states during client-server communication. Developers may find this list useful when writing their own file transfer procedures using low-level TCP commands.

110	Restart marker reply. In this case, the text is exact and not left to the particular implementation; it must read: MARK yyyy = mmmm. Where yyyy is User-process data stream marker, and mmmm server's equivalent marker (note the spaces between markers and "=").
120	Service ready in nnn minutes.
125	Data connection already open; transfer starting.
150	File status okay; about to open data connection.
200	Command okay.
202	Command not implemented, superfluous at this site.
211	System status, or system help reply.
212	Directory status.
213	File status.
214	Help message on how to use the server or the meaning of a particular non-standard command. This reply is useful only to the human user.
215	NAME system type. Where NAME is an official system name from the list in the Assigned Numbers document.
220	Service ready for new user.
221	Service closing control connection. Logged out if appropriate.
225	Data connection open; no transfer in progress.
226	Closing data connection. Requested file action successful (e.g., file transfer or file abort).
227	Entering Passive Mode (h1,h2,h3,h4,p1,p2).
230	User logged in, proceed.
250	Requested file action okay, completed.
257	"PATHNAME" created.
331	User name okay, need password.
332	Need account for login.
350	Requested file action pending further information.
421	Service not available, closing control connection. This may be a reply to any command if the service knows it must shut down.
425	Can't open data connection.
426	Connection closed; transfer aborted.
450	Requested file action not taken. File unavailable (file busy).
451	Requested action aborted: local error in processing.
452	Requested action not taken. Insufficient storage space in system.
500	Syntax error, command unrecognized. This may include errors such as command line too long.
501	Syntax error in parameters or arguments.
502	Command not implemented.
503	Bad sequence of commands.
504	Command not implemented for that parameter.
530	Not logged in.
532	Need account for storing files.
550	Requested action not taken. File unavailable (e.g., file not found, no access).
551	Requested action aborted: page type unknown.
552	Requested file action aborted. Exceeded storage allocation (for current directory or dataset).
553	Requested action not taken. File name not allowed.

Appendix D, Additional Information...

The references below contain WWW (World Wide Web) pointers to additional sources of information related to the internet protocols. Web documents may be accessed via programs like Netscape or Internet Explorer.

http://www.internic.net/: To understand what a Domain Name is and what you have to do to register one.

http://www.ietf.org/: Internet Engineering Task Force (IETF) site.

http://www.rfc-editor.org/: To understand what an RFC is and to search for RFCs and sites related to the RFC series (http://www.rfc-editor.org/rfc.html).

ftp://ftp.isi.edu/in-notes/rfc821.txt: Simple Mail Transfer Protocol -- RFC 821.

http://www.w3c.org/: All you need to know about the World Wide Web.

http://www.imap.org: Site reserved for IMAP protocol. You will find all the useful information concerning this protocol there.

Appendix E, Testing mail by dumping in a local file

Testing and debugging code that sends emails could be a pain; when an email is not received correctly, the source of the problem could be the network configuration, the provider, the client software, and so on.

To help you in this case, we added the ability to dump email in a local file instead of sending it. With this feature, you just need to modify the resulting file slighlty to create an EML file that can display the results in MS Outlook. You can also include email files in unit-testing procedures.

Code for the local test

You can run the local code:

```
  $err:=SMTP_SetPrefs(0;15;0) // Body: UTF-8  &  QuotedPrintable, Header: UTF-8  &  Base64
  $err:=SMTP_Charset(1;1) // Apply setting to header and body (Body: UTF-8  &  QuotedPrintable
  &  Header: UTF-8  &  Base64)

   //$hostName:="smtp.gmail.com"  // This line sends it over the network using smtp.gmail.com
  $hostName:="file:C:\\Users\\MyWinUser\\Desktop\\test.txt" // This line doesn't send the email,
 instead it saves the bytes in a file you can monitor
  $msgTo:="mail.to@gmail.com"
  $msgFrom:="mail.sender@gmail.com"

  $mailSubject:="テストメール(v17 4372) " //test using extended characters
  $mailBody:="日本語で終わる"
  $err:=SMTP_QuickSend($hostName;$msgFrom;$msgTo;$mailSubject;$mailBody;0;0;$msgFrom;"password")
```

This produces the following .txt file:

```
<mail.sender@gmail.com>
<mail.to@gmail.com>

Mime-Version: 1.0
Content-Type: text/plain;charset="utf-8"
Content-Transfer-Encoding: quoted-printable
Date: Fri, 08 Jul 2016 16:45:24 +0200
To: mail.to@gmail.com
From: mail.sender@gmail.com
Subject: =?utf-8?B?44OG44K544OI44Oh44O844Or77yIdjE3IDQzNzIpIA==?=

=E6=97=A5=E6=9C=AC=E8=AA=9E=E3=81=A7=E7=B5=82=E3=82=8F=E3=82=8B
```

If you want to open this file as a standard MS Outlook email:

1. Delete all lines before "Mime-Version: 1.0":

   ```
   Mime-Version: 1.0
   Content-Type: text/plain;charset="utf-8"
   Content-Transfer-Encoding: quoted-printable
   Date: Fri, 08 Jul 2016 16:45:24 +0200
   To: mail.to@gmail.com
   From: mail.sender@gmail.com
   Subject: =?utf-8?B?44OG44K544OI44Oh44O844Or77yIdjE3IDQzNzIpIA==?=

   =E6=97=A5=E6=9C=AC=E8=AA=9E=E3=81=A7=E7=B5=82=E3=82=8F=E3=82=8B
   ```

2. Save this file with the ".eml" extension, for example "test.eml".
3. Double-click the file and you will see the email in MS Outlook just as if you received it from your mail server.

Timestamp considerations

To comply with unit-testing mechanisms, when using **SMTP_QuickSend** with an output file, the Date header is always the following string:

```
Date: Fri, 08 Jul 2016 16:45:24 +0200
```

In this case, date comparisons will not fail in unit tests.

Note: When you use a real host (such as smtp.gmail.com), the date header gets replaced by an actual timestamp.

If you want to have an actual timestamp in your test file, you can use this feature with **SMTP_Send**. In this case, you can use the **SMTP_Date** command and then provide an actual Date header with **SMTP_Send**.