Sending and Receiving E-Mails in Unicode

By Keisuke Miyako, 4D Japan.

Technical Note 09-17

Table of Contents

Abstract

Starting with 4D Internet Commands for 4D v11 SQL, the developer can now send and receive e-mails of various languages with relative ease; this is because 4D v11 SQL works in Unicode and has the ability of converting to and from various character sets irrespective of the system language. This Tech Note describes the basic concepts related to the e-mail protocol while pointing out some limitations that one may possibly face when attempting to send and receive e-mails in Unicode with 4D Internet Commands. A sample component is provided as a sample solution.

Introduction

One of the key features of 4D Internet Commands is its set of high-level commands that lets the developer send and receive e-mail in the 4D environment. Originally, it was designed to handle the US-ASCII and ISO-8859-1 encodings only, used mainly in the United States and Western Europe. Later it was enhanced to automatically support ISO-2022-JP and Shift-JIS when used in a Japanese environment. In order to handle e-mails composed in other character sets, the developer had to encode or decode the message header and body themselves.

Starting with 4D v11 SQL, the developer can now send and receive e-mails using various character sets with relative ease, including koi8-r (used in countries that belonged to the former Soviet Union), ISO-8859-15 (which includes the euro symbol), EUC-KR (used for Korean), Big-5 (used in Taiwan), GB2312 (used in mainland China), as well as UTF-8 (Unicode). In previous versions of 4D, one could only do so by resorting to tricks such as changing the system language or giving the form object used to edit or display the message a keyboard layout setting that would override the system language. This is no longer the case with 4D v11 SQL.

In an increasingly internationalized world, it is not unusual for one to be asked to process e-mail messages in various different languages. 4D Internet Commands does have some useful features that automatically perform encoding and decoding, but the developer may have to take command over the situation under certain circumstances. A good understanding of the e-mail protocol and 4D Internet Commands comes in handy to that end.

Why a message needs to be encoded for transfer

Described below are some basic features of the e-mail protocol and the historic background behind its specification. The reader may skip this section if already familiar with such information.

The 7-bit legacy

When the e-mail protocol was first invented, the objective was to send and receive electronic messages composed in US-ASCII, a set of 127 characters each 7 bits in size. As such, an e-mail message was expected to contain only US-ASCII characters and an e-mail server was only required to process 7-bit data. Although a modern e-mail message may seemingly contain extended

characters outside the US-ASCII range or even images and binaries for that matter, the source of such an e-mail message still contains only 7-bit characters that conform to the original e-mail protocol. To overcome the 7-bit limitation, an extension to the original e-mail format called MIME (Multipurpose Internet Mail Extensions) was introduced, in which one or more of the following encoding techniques are used.

Quoted Printable

For a typical e-mail body or header where the message is written in English or a language used in Western Europe, the letters used are predominantly US-ASCII characters with the occasional exception of diacritical characters or some special symbols. In such cases, it makes sense to transmit the basic US-ASCII characters directly while escaping the occasional 8-bit characters by prefixing them with the '=' sign followed by its hexadecimal character code. This way, most of the message will remain untouched and easily readable (or, printable). Incidentally the '=' sign will also be escaped. This method of encoding is known as Quoted Printable and used for messages that mainly use the Latin alphabet.

base64

For languages that rarely if ever use the Latin alphabet (for example Chinese, Korean or Japanese) there is not much meaning in applying the Quoted Printable convention since a large part of the original message will have to be quoted and the message will contain little if any content that is printable. In such cases, it is more usual (though not mandatory) to use a different encoding method known as base64, which will replace every three bytes in the original data with a set of four characters, thus increasing the size by 33% but ensuring that only one of the 64 'safe' characters will ever appear in the resulting text. This form of encoding can be applied to any kind of data including images and binary attachments.

7bit

If the message contains only basic ASCII characters, as with the US-ASCII encoding, it can be directly sent as e-mail in its original format. Some types of encoding like ISO-2022 are also limited to the basic US-ASCII characters and suitable for direct 7-bit transmission.

8bit

Some people choose to send 8-bit encoded data (ISO-8859-1, for example) directly without using either the Quoted Printable or base64 encoding. This would work, provided the mail servers used to relay the message are all 8-bit compliant. If one of them happens to be an old 7-bit mail server, the message could unfortunately get corrupt. You should not be tempted to send a message in this format, though you should be prepared to occasionally receive one in such.

Format of an e-mail message

IMF

The formal specification of the original Internet Message Format without the MIME extension is defined as RFC-5322. In essence it is required that an e-mail contains only US-ASCII characters, separated into lines by a CR (ASCII 13) and LF (ASCII 10) character pair whose lengths must not exceed 1000 characters including the terminating sequence. A message consists of a header section, an empty line (two sets of CR-LF) and a body section. The header section can have any number of lines that begin with a field name, followed by a colon, followed by a field body. The field body can be "folded" to allow values that exceed the 1000 character per line limitation. The body section is a series of US-ASCII lines that contains no more than 1000 characters each.

MIME

The MIME extension sets conventions, not withstanding the above specification, to transmit characters outside the US-ASCII range as an e-mail message. It also specifies rules that allow an e-mail message to have several bodies, known as multipart contents. Other extensions provide instructions as to how rare but nevertheless possible exceptions should be handled. The MIME specification is defined as RFC-2045, RFC-2046, RFC-2047, RFC-2049, RFC-4288 and RFC-4289.

How the Header is encoded

Conforming to the RFC-1342 standard, the header value should be encoded in the following format should it contain any characters outside the US-ASCII range.

```
=?<character_set_name>?<B_or_Q>?<encoded_text>?=
```

The encoding convention can be base64 or Quoted Printable, depending on how heavily the text uses basic US-ASCII characters. Several instances of the above sequence may appear in a single header, each with distinct directives.

How the Body is encoded

The encoding convention used for the body section is specified in the message header. More specifically, the 'Content-Type' header tells which character set is used in its 'charset' attribute, and the 'Content-Transfer-Encoding' header tells which of the following methods is used to encode the text; 7bit, 8bit, base64 or Quoted Printable.

The encoding capabilities of 4D Internet Commands

4D Internet Commands has a built-in feature designed to automatically encode both the header and body sections of an outgoing e-mail message. Two commands

are used in particular, *SMTP_Setprefs* and *SMTP_Charset.* The former is used to declare which type of encoding is to be used, and the latter is used to define the scope of such conversion (Header, Body or both).

Limitations with the 'Setprefs' and 'Charset' commands

4D Internet Commands (as of 4D v11 SQL Release 3) is a pre-version 11 plugin that does NOT handle Unicode natively. It receives and returns text in the legacy TEXT format, which is not only limited in its size (32000) but also its range of expression, in the sense that a non-Unicode (ANSI) encoding that corresponds to the system language is used. In other words, any characters outside the basic US-ASCII range can be lost or corrupt if the source Unicode TEXT cannot be fully converted to the legacy encoding that corresponds to the system language.

To give an example, say 4D is running on an English system, and a TEXT variable (which is Unicode) contained some Japanese characters. To pass this variable to 4D Internet Commands would force a Unicode-to-Latin (ISO-8859-1) conversion on this TEXT, because ISO-8859-1 is the preferred legacy encoding for English, thus corrupting all data except for the common US-ASCII characters.

If the system environment happened to be Japanese, then a Unicode-to-Japanese (Shift-JIS) conversion would have taken place instead. In this case the data would be intact, so the developer can call *SMTP_Setprefs* in conjunction with *SMTP_Charset* and ask the plugin to perform the necessary encoding (ISO-2022-JP, which effectively is like a 7-bit encoding of the Shift-JIS character set).

The same applies to the first example. If the variable contained only Latin characters, and the system language was English, then the entire TEXT content would have safely passed the plugin API's conversion. The developer can then use SMTP commands to automatically handle the necessary encoding (ISO-8859-1, usually Quoted Printable).

Note on the 'Setprefs' and 'UTF-8' settings

Starting with 4D v11 SQL, 4D Internet Commands offer the 'UTF-8 Quoted Printable' and 'UTF-8 base64' options with *SMTP_Setprefs*. However the simple fact that the plugin does not receive text in Unicode but rather some legacy encoding means that these options are fundamentally restricted. More specifically, the plugin will treat non-Unicode text as if they were Unicode, thus producing an erroneous output. The encoded result text would be valid for the US-ASCII range only.

To summarize, the developer can only resort to the built-in encoding capabilities of 4D Internet Commands if a) the system language is Western European and the message is ISO-8859-1/US-ASCII compatible, or b) the system language is Japanese and the message is ISO-2022-JP/US-ASCII compatible. For all other cases one would have to create a custom encoding method.

The decoding capabilities of 4D Internet Commands

4D Internet Commands also has a built-in feature to automatically decode both the header and body sections of an incoming e-mail message or a downloaded message. As with outgoing messages, the *POP3_Charset*/*MSG_Charset* commands are used to define the scope of such conversion. Again, either ISO-8859-1 (Latin) or ISO-2022-JP (Japanese) is supported, depending on the system language. For all other cases the developer would have to create a custom decoding method.

Encoding examples

base64

Encoding in base64 is pretty straightforward, thanks to the command **ENCODE** introduced in 4D 2004. One can perform such conversion as shown in the following example.

```
(encode base64)

C_TEXT($1;$2;$0)

If (Count parameters=2)

   C_BLOB($base64_x)
     `convert the unicode text to a blob of the target encoding
   $base64_x:=encode text ($1;$2)
   `perform base64 encoding
   ENCODE($base64_x)
     `now the data contains 'safe' characters, so revert to text
   $0:=Convert to text($base64_x;"ISO-8859-1")

End if
```

Quoted Printable

Unlike base64 there is no native command that can encode text in Quoted Printable form. One can perhaps use a loop routine like the one shown below.

```
(encode quoted printable)

C_TEXT($1;$2;$0)

If (Count parameters=2)

   $character_set_t:=$2

   C_BLOB($quoted_printable_x)
     `convert the unicode text to a blob of the target encoding
   $quoted_printable_x:=encode text ($1;$2)

   C_TEXT($quoted_printable_t)
     `repeat with every byte
   For ($i;0;BLOB size($quoted_printable_x)-1)

     Case of
```

```
          : ($quoted_printable_x{$i}=0x0020)
                  `the space character
               $quoted_printable_t:=$quoted_printable_t+"=20"

          : ($quoted_printable_x{$i}=0x003D)
                  `the equals sign
               $quoted_printable_t:=$quoted_printable_t+"=3D"

          : ($quoted_printable_x{$i}<0x007F)
                  `all other US-ASCII characters

   $quoted_printable_t:=$quoted_printable_t+Char($quoted_printable_x{$i})

   Else
     `extended characters
   $quoted_printable_t:=$quoted_printable_t+"="+longint to hex
($quoted_printable_x{$i})

          End case

   End for

   $0:=$quoted_printable_t

End if
```

Once such methods are made available, variants of them for the header section can also be created. For example, to create a header field (RFC-1342) using base64, one could write some code like the one shown below.

```
(header value base64)

C_TEXT($1;$2;$0)

If (Count parameters=2)

   $encoded_header_value_t:=encode base64 ($1;$2)

   $0:=Choose($encoded_header_value_t="";$encoded_header_value_t;"=?"+$2+"?B?"+
$encoded_header_value_t+"?=")

End if
```

A Quoted Printable type header can also be created in similar fashion.

```
(header value quoted printable)

C_TEXT($1;$2;$0)

If (Count parameters=2)

   $encoded_header_value_t:=encode quoted printable ($1;$2)

   $0:=Choose($encoded_header_value_t="";$encoded_header_value_t;"=?"+$2+"?Q?"+
$encoded_header_value_t+"?=")

End if
```

Decoding examples

base64

Decoding in base64 is fairly straightforward, thanks to the command **DECODE**.

```
(decode base64)

C_TEXT($1;$2;$0)

If (Count parameters=2)
      `we assume the source text contains only 'safe' characters
   C_BLOB($base64_x)
   CONVERT FROM TEXT($1;"ISO-8859-1";$base64_x)
      `perform base64 decoding
   DECODE($base64_x)
      `convert the decoded blob of the source encoding to unicode
   $0:=decode text ($base64_x;$2)

End if
```

Quoted Printable

Decoding Quoted Printable data can be quite tricky. Here is an example solution. Notice how we take advantage of the new **Match regex** function.

```
(decode quoted printable)

C_TEXT($1;$2;$0)

If (Count parameters=2)

   $i:=1

   ARRAY LONGINT($match_positions_al;0)
   ARRAY LONGINT($match_lengths_al;0)

   C_BLOB($encoded_text_x)
      `match '=xx' ignoring the case
   While (Match regex("(.*?)=((?i)[a-f0-
9]{2})";$1;$i;$match_positions_al;$match_lengths_al))
            `collect the prefix plain text, if any
         CONVERT FROM TEXT(Replace
string(Substring($1;$match_positions_al{1};$match_lengths_al{1});"_";"");"UTF-
8";$plain_text_x)
         COPY BLOB($plain_text_x;$encoded_text_x;0;BLOB
size($encoded_text_x);BLOB size($plain_text_x))
            `we are only interested in the insignificant bytes, use big endian
         INTEGER TO BLOB(longint from hex
(Substring($1;$match_positions_al{2};$match_lengths_al{2}));$escaped_character_
x;PC byte ordering )
         COPY BLOB($escaped_character_x;$encoded_text_x;0;BLOB
size($encoded_text_x);1)
         $i:=$match_positions_al{2}+$match_lengths_al{2}

   End while
      `append the suffix plain text, if any
   CONVERT FROM TEXT(Substring($1;$i;Length($1));"ISO-8859-1";$plain_text_x)
```

```
   COPY BLOB($plain_text_x;$encoded_text_x;0;BLOB size($encoded_text_x);BLOB
size($plain_text_x))
      `convert the decoded blob of the source encoding to unicode
   $0:=decode text ($encoded_text_x;$2)

End if
```

Once such methods are made available, we can create a wrapper routine to generically decode e-mail header fields (base64 or quoted printable).

```
(header value resolve encoding)

C_TEXT($1;$0)

If (Count parameters=1)

   $i:=1

   ARRAY LONGINT($match_positions_al;0)
   ARRAY LONGINT($match_lengths_al;0)

   C_BLOB($base64_x)

   $subject_t:=""
      `the line can contain multiple instances of an RFC-1342 sequence
   While (Match
regex("(?s)(.*?)=\\?(.+?)\\?(B|Q)\\?(.*?)(\\?=)(\\s*)";$1;$i;$match_positions_a
l;$match_lengths_al))
            `collect the prefix plain text, if any
   $subject_t:=$subject_t+Substring($1;$match_positions_al{1};$match_lengths_al
{1})
            `capture the encoding name

   $character_set_t:=Substring($1;$match_positions_al{2};$match_lengths_al{2})
            `can be either 'B' (base64) or 'Q' quoted printable

   $character_encoding_t:=Substring($1;$match_positions_al{3};$match_lengths_al
{3})

          Case of
            : ($character_encoding_t="B")

                        $subject_t:=$subject_t+decode base64
(Substring($1;$match_positions_al{4};$match_lengths_al{4});$character_set_t)

                : ($character_encoding_t="Q")

                        $subject_t:=$subject_t+decode quoted printable
(Substring($1;$match_positions_al{4};$match_lengths_al{4});$character_set_t)

          End case

          $i:=$match_positions_al{6}+$match_lengths_al{6}

   End while
      `append the suffix plain text, if any
   $0:=Choose(($1#"") &
($subject_t="");$1;$subject_t+Substring($1;$i;Length($1)))

End if
```

Text conversion capabilities of 4D v11 SQL

A TEXT field or variable in 4D v11 SQL is always Unicode, and 'text' in every other format can only exist as a BLOB. The commands **Convert to text** and **CONVERT FROM TEXT** can be used to convert encodings to and from Unicode respectively.

Unsupported encodings

As of 11.3, several character encodings widely used for e-mail transmission are unfortunately not supported with the standard conversion commands. Among them are ISO-8859-15 (which contains the euro symbol; now supported in 11.4), ISO-2022-JP (Japanese), GB18030 (Chinese) and several EUC encodings (UNIX). The sample provided with this Tech Note comes with a simple plugin that expands the range of supported encodings, used in the wrapper methods '*encode text*' and '*decode text*'. One may expand or restrict the range of encodings that bypass the standard functions by modifying these methods.

Widely accepted encodings for daily e-mail exchange

The standard e-mail character set for Simplified Chinese used in Mainland China is EUC-CN, also known as GB2312. There is also a 7-bit version called HZ GB2312. Unlike Japanese, the 7-bit convention ISO-2022-CN is hardly ever used. The current official character set is GB18030, which covers Simplified and Traditional Chinese, Korean, Japanese as well as various characters used by ethnic minorities. GB18030 is similar to but not quite the same as Unicode.

The standard e-mail character set for Japanese is the 7-bit convention ISO-2022-JP. The enhanced version ISO-2022-JP-2 can include characters from ISO-8859-1 (Latin), EUC-KR (Korean), GB2312 (Simplified Chinese) and ISO-8859-7 (Greek). Sending e-mail in Shift-JIS (base64) is generally considered to be bad practice, though most modern systems will accept such messages without a problem. Note that the use of half-width Kana characters is not compatible with ISO-2022-JP.

The standard e-mail character set for Korean is EUC-KR. Unlike Japanese, the 7-bit convention ISO-2022-KR is hardly ever used. The Democratic People's Republic of Korea (North Korea) does have its own standards but EUC-KR is more popular.

The standard e-mail character set for Russian is KOI8-R. Windows-1251 also happens to be popular. ISO-8859-5 is hardly ever used.

The standard e-mail character set in Western Europe is ISO-8859-1. Its variants Windows-1252 and Mac Roman are slightly different in that they replace several control characters with symbols. Such control characters are mostly not welcome in Internet communication, for which reason many applications tend to actually use Windows-1252 (though they may claim to use ISO-8859-1) instead.

The standard e-mail character set for Arabic is indeed Unicode. ISO-8859-6 ishardly ever used.

Dissecting multipart content E-mails

The MIME specification defines how multiple body contents can be included in a single e-mail message. The extension allows various types of files to be embedded as attachments. It also enables the author to use rich text such as HTML to compose the body text instead of plain text. A MIME part itself can have several body contents, effectively creating a multi layered structure of contents. In any case, each part has its own set of header fields that describe its nature and identity.

A multi part content is segmented by boundaries as declared in the header. One may dissect such messages into individual parts by looking for the specific boundary, as shown in the example below. Note how the method makes a recursive call to itself, since a multi part content can have any number of cascaded levels.

```
(eml MIME parts)

C_TEXT($1;$4)
C_POINTER($2;$3)

$count_parameters_l:=Count parameters

Case of
    : ($count_parameters_l=3)
            `this is our first execution
          If (Not(Nil($2))) & (Not(Nil($3)))

            If (Type($2->)=Text array ) & (Type($3->)=Text array )

                  ARRAY TEXT($2->;0)
                  ARRAY TEXT($3->;0)
                    `looking at the base level header
                  $data:=eml header ($1)
                    `retrive the base boundary, if exists
                  $boundary_t:=eml boundary ($data)

                  If ($boundary_t="")  `single part e-mail

                        APPEND TO ARRAY($2->;eml body ($1))
                        APPEND TO ARRAY($3->;eml MIME ($data))

                  Else    `multi part e-mail
                         `recursive call
                       eml MIME parts (eml body ($1);$2;$3;$boundary_t)

                  End if

            End if

          End if
            `this is a recursive call
    : ($count_parameters_l=4)

          If (Not(Nil($2))) & (Not(Nil($3)))
```

```
                If (Type($2->)=Text array ) & (Type($3->)=Text array )

                      ARRAY TEXT($parts_at;0)
                        `get the MIME body, extracted by the boundary ($4)
                      $count_parts_l:=eml body parts ($1;$4;->$parts_at)

                      For ($i;1;Size of array($parts_at))
                              `look for a lower layer
                            $boundary_t:=eml boundary ($parts_at{$i})

                        If ($boundary_t="")  `bottom level

                          APPEND TO ARRAY($2->;$parts_at{$i})
                          APPEND TO ARRAY($3->;eml MIME ($parts_at{$i}))

                        Else   `more parts, dig deeper with a recursive call

                          eml MIME parts ($parts_at{$i};$2;$3;$boundary_t)

                        End if

                      End for

                End if

          End if

End case
```

The 4D Email Tools component

This sample structure, which can also be embedded as a component, features various utilities for sending and receiving e-mails in Unicode, both for research and practical purposes. Forms for editing and retrieving messages are provided. The code uses, though is not dependent on, a custom plugin to perform conversion to and from various character sets. 4D Internet Commands is necessary for operation.

Example code

A set of example code is included as a macro. Open an empty method, press 'H' and trigger a type ahead to call the macro.

Sending messages

A collection of high-level methods named '*eml send <character_set_name>*' is provided to simplify the procedure of converting, encoding and sending messages. Supported character sets are, ISO-2022-JP, Shift-JIS, GB2312, Gb18030, Big5, ISO-2022-KR, EUC-KR, ISO-8859-1, ISO-8859-15 and UTF-8.All are simple wrappers of a lower routine and other variants can easily be created if necessary.

The list of parameters is as follows.

- $1 Host name (e.g. "4d-japan.com")

- $2 From (e.g. “miy@4d-japan.com” or “me <“miy@4d-japan.com” >”)
- $3 Subject (max 32000, Unicode TEXT)
- $4 Body (max 2GB, Unicode TEXT)
- $5 To (pointer to array of Unicode TEXT)
- $6 Cc (pointer to array of Unicode TEXT, optional)
- $7 Bcc (pointer to array of Unicode TEXT, optional)
- $8 Attachment file paths (pointer to array of Unicode TEXT, optional)
- $0 Result (Boolean, True if successful)

Reading received messages

A set of commands to extract various parts of a downloaded e-mail message is provided. Regular expressions are used to retrieve the charset, encoding, date, time, subject and sender information. The sender’s name, if provided, is decoded. The subject is decoded and returned as Unicode. The time and date are converted to TIME and DATE variables. Attachments are returned as an array of PICTURE variables, which are in fact BLOB containers that can each be reverted to BLOB if necessary (a feature introduced in 11.2).

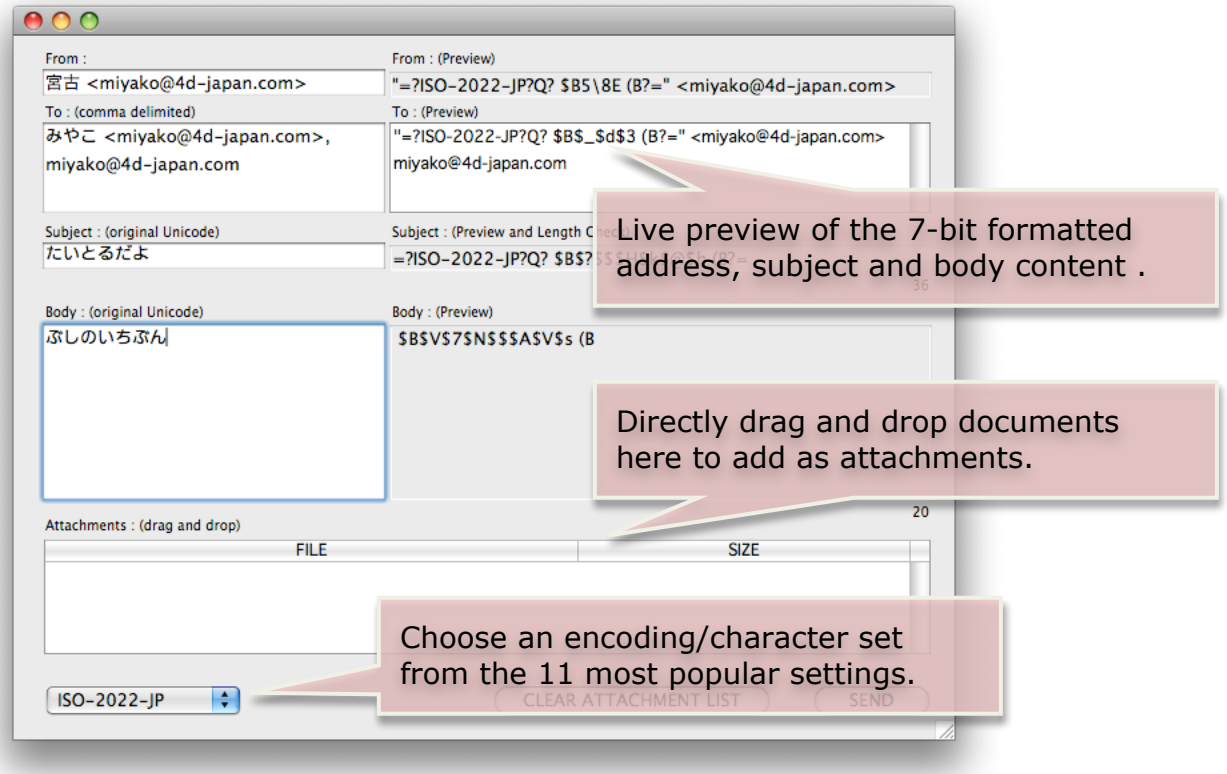
Receiving messages

To retrieve messages from a POP server, first create a mail account structure by calling the method “*mail account*”. This method uses the version 11 feature to create key-value pairs (also known as dictionaries) in a hierarchical list item. The list is converted to BLOB, which we will use as an abstract representation of an e-mail account.

Two methods are provided, one that simply retrieves from the POP server a list of IDs and another that fetches more detailed information.

Editing an outgoing message

A simple form to let one edit and preview an outgoing message is provided. Execute the method “*_send*” or directly run the form in Design Mode. Notice how regular expressions are used to detect whether an e-mail address is plain or RFC structured. A length check on the subject field is performed, since 4D Internet Commands can only accept up to 32000 characters for each header field. The body text may be of any length but the length count is displayed for reference. Any number of attachments can be directly dragged and dropped to the attachments list box.

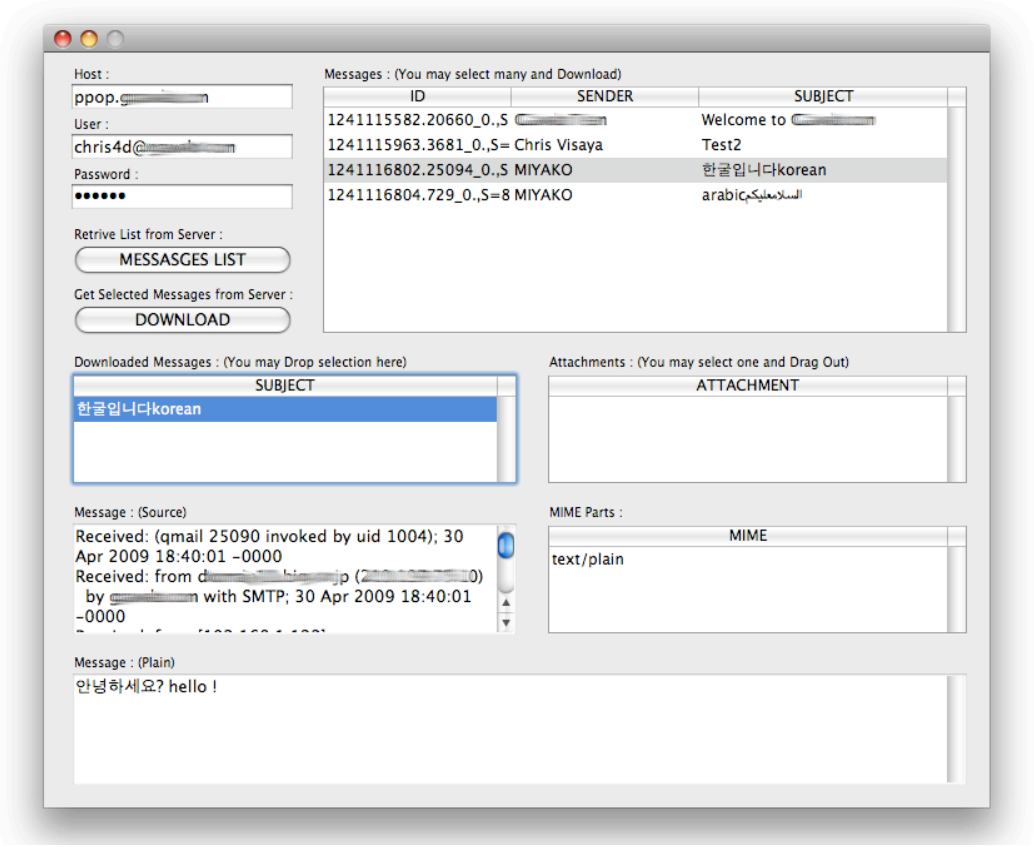


Viewing received messages

A simple form to let one view the source and decoded body of e-mail messages is provided. Execute the method "*_receive*" or directly run the form in Design Mode. Notice how the id, sender and subject of messages are retrieved from the server asynchronously. This is done by taking advantage of the version 11 feature of putting a pointer to a process variable in an inter-process variable, and updating its deferred value from another process. To test this,

If the "from" header is RFC structured, the sender's name is decoded and returned in Unicode. If not, the plain e-mail address is given. The subject is also decoded and returned in Unicode. The body of one or more messages can be selected and downloaded. If the message is MIME, a list of its MIME parts and their MIME types is displayed. For parts whose type is "text/plain", its content can be decoded and displayed in Unicode.

Last, be aware that currently this form is set up to only accept POP access (as opposed to IMAP), and no SSL enabled.



Conclusion

This Tech Note described the basic concepts related to the e-mail protocol and its implementation in 4D Internet Commands, while pointing out some limitations that one may possibly face when attempting to send and receive e-mails in Unicode with 4D Internet Commands. A component was presented as a sample solution.