Network Working Group   C. Newman
Internet Draft: IMAP URL Scheme   Innosoft
Document: draft-newman-url-imap-09.txt   May 1997
                                          Expires in six months

                          IMAP URL Scheme

Status of this memo

Abstract

    IMAP [IMAP4] is a rich protocol for accessing remote message
    stores. It provides an ideal mechanism for accessing public
    mailing list archives as well as private and shared message stores.
    This document defines a URL scheme for referencing objects on an
    IMAP server.

1. Conventions used in this document

    The key words "MUST", "MUST NOT", "SHOULD", "SHOULD NOT", and "MAY"
    in this document are to be interpreted as defined in "Key words for
    use in RFCs to Indicate Requirement Levels" [KEYWORDS].

2. **IMAP scheme**

The IMAP URL scheme is used to designate IMAP servers, mailboxes, messages, MIME bodies [MIME], and search programs on Internet hosts accessible using the IMAP protocol.

The IMAP URL follows the common Internet scheme syntax as defined in RFC 1738 [BASIC-URL] except that clear text passwords are not permitted. If :<port> is omitted, the port defaults to 143.

An IMAP URL takes one of the following forms:

```
imap://<iserver>/
imap://<iserver>/<enc_list_mailbox>;TYPE=<list_type>
imap://<iserver>/<enc_mailbox>[uidvalidity][?<enc_search>]
imap://<iserver>/<enc_mailbox>[uidvalidity]<iuid>[isection]
```

The first form is used to refer to an IMAP server, the second form refers to a list of mailboxes, the third form refers to the contents of a mailbox or a set of messages resulting from a search, and the final form refers to a specific message or message part. Note that the syntax here is informal. The authoritative formal syntax for IMAP URLs is defined in section 11.

3. **IMAP User Name and Authentication Mechanism**

A user name and/or authentication mechanism may be supplied. They are used in the "LOGIN" or "AUTHENTICATE" commands after making the connection to the IMAP server. If no user name or authentication mechanism is supplied, the user name "anonymous" is used with the "LOGIN" command and the password is supplied as the Internet e-mail address of the end user accessing the resource. If the URL doesn't supply a user name, the program interpreting the IMAP URL SHOULD request one from the user if necessary.

An authentication mechanism can be expressed by adding ";AUTH=<enc_auth_type>" to the end of the user name. When such an <enc_auth_type> is indicated, the client SHOULD request appropriate credentials from that mechanism and use the "AUTHENTICATE" command instead of the "LOGIN" command. If no user name is specified, one SHOULD be obtained from the mechanism or requested from the user as appropriate.

The string ";AUTH=*" indicates that the client SHOULD select an appropriate authentication mechanism. It MAY use any mechanism listed in the CAPABILITY command or use an out of band security service resulting in a PREAUTH connection. If no user name is

specified and no appropriate authentication mechanisms are
available, the client SHOULD fall back to anonymous login as
described above. This allows a URL which grants read-write access
to authorized users, and read-only anonymous access to other users.

If a user name is included with no authentication mechanism, then
";AUTH=*" is assumed.

Since URLs can easily come from untrusted sources, care must be
taken when resolving a URL which requires or requests any sort of
authentication. If authentication credentials are supplied to the
wrong server, it may compromise the security of the user's account.
The program resolving the URL should make sure it meets at least
one of the following criteria in this case:

(1) The URL comes from a trusted source, such as a referral server
which the client has validated and trusts according to site policy.
Note that user entry of the URL may or may not count as a trusted
source, depending on the experience level of the user and site
policy.
(2) Explicit local site policy permits the client to connect to the
server in the URL. For example, if the client knows the site
domain name, site policy may dictate that any hostname ending in
that domain is trusted.
(3) The user confirms that connecting to that domain name with the
specified credentials and/or mechanism is permitted.
(4) A mechanism is used which validates the server before passing
potentially compromising client credentials.
(5) An authentication mechanism is used which will not reveal
information to the server which could be used to compromise future
connections.

URLs which do not include a user name must be treated with extra
care, since they are more likely to compromise the user's primary
account. A URL containing ";AUTH=*" must also be treated with
extra care since it might fall back on a weaker security mechanism.
Finally, clients are discouraged from using a plain text password
as a fallback with ";AUTH=*" unless the connection has strong
encryption (e.g. a key length of greater than 56 bits).

Note that if unsafe or reserved characters such as " " or ";" are
present in the user name or authentication mechanism, they MUST be
encoded as described in RFC 1738 [BASIC-URL].

[4](#). IMAP server

An IMAP URL referring to an IMAP server has the following form:

    imap://<iserver>/

A program interpreting this URL would issue the standard set of
commands it uses to present a view of the contents of an IMAP
server. This is likely to be semanticly equivalent to one of the
following URLs:

    imap://<iserver>/;TYPE=LIST
    imap://<iserver>/;TYPE=LSUB

The program interpreting this URL SHOULD use the LSUB form if it
supports mailbox subscriptions.

[5](#). Lists of mailboxes

An IMAP URL referring to a list of mailboxes has the following
form:

    imap://<iserver>/<enc_list_mailbox>;TYPE=<list_type>

The <list_type> may be either "LIST" or "LSUB", and is case
insensitive. The field ";TYPE=<list_type>" MUST be included.

The <enc_list_mailbox> is any argument suitable for the
list_mailbox field of the IMAP [IMAP4] LIST or LSUB commands. The
field <enc_list_mailbox> may be omitted, in which case the program
interpreting the IMAP URL may use "*" or "%" as the
<enc_list_mailbox>. The program SHOULD use "%" if it supports a
hierarchical view, otherwise it SHOULD use "*".

Note that if unsafe or reserved characters such as " " or "%" are
present in <enc_list_mailbox> they MUST be encoded as described in
RFC 1738 [BASIC-URL]. If the character "/" is present in
enc_list_mailbox, it SHOULD NOT be encoded.

[6](#). Lists of messages

An IMAP URL referring to a list of messages has the following form:

    imap://<iserver>/<enc_mailbox>[uidvalidity][?<enc_search>]

The <enc_mailbox> field is used as the argument to the IMAP4

"SELECT" command. Note that if unsafe or reserved characters such
as " ", ";", or "?" are present in <enc_mailbox> they MUST be
encoded as described in RFC 1738 [BASIC-URL]. If the character "/"
is present in enc_mailbox, it SHOULD NOT be encoded.

The [uidvalidity] field is optional. If it is present, it MUST be
the argument to the IMAP4 UIDVALIDITY status response at the time
the URL was created. This SHOULD be used by the program
interpreting the IMAP URL to determine if the URL is stale.

The [?<enc_search>] field is optional. If it is not present, the
contents of the mailbox SHOULD be presented by the program
interpreting the URL. If it is present, it SHOULD be used as the
arguments following an IMAP4 SEARCH command with unsafe characters
such as " " (which are likely to be present in the <enc_search>)
encoded as described in RFC 1738 [BASIC-URL].

7. A specific message or message part

An IMAP URL referring to a specific message or message part has the
following form:

    imap://<iserver>/<enc_mailbox>[uidvalidity]<iuid>[isection]

The <enc_mailbox> and [uidvalidity] are as defined above.

If [uidvalidity] is present in this form, it SHOULD be used by the
program interpreting the URL to determine if the URL is stale.

The <iuid> refers to an IMAP4 message UID, and SHOULD be used as
the <set> argument to the IMAP4 "UID FETCH" command.

The [isection] field is optional. If not present, the URL refers
to the entire Internet message as returned by the IMAP command "UID
FETCH <uid> BODY.PEEK[]". If present, the URL refers to the object
returned by a "UID FETCH <uid> BODY.PEEK[<section>]" command. The
type of the object may be determined with a "UID FETCH <uid>
BODYSTRUCTURE" command and locating the appropriate part in the
resulting BODYSTRUCTURE. Note that unsafe characters in [isection]
MUST be encoded as described in [BASIC-URL].

8. Relative IMAP URLs

Relative IMAP URLs are permitted and are resolved according to the
rules defined in RFC 1808 [REL-URL] with one exception. In IMAP
URLs, parameters are treated as part of the normal path with

respect to relative URL resolution. This is believed to be the
behavior of the installed base and is likely to be documented in a
future revision of the relative URL specification.

The following observations are also important:

The <iauth> grammar element is considered part of the user name for
purposes of resolving relative IMAP URLs. This means that unless a
new login/server specification is included in the relative URL, the
authentication mechanism is inherited from a base IMAP URL.

URLs always use "/" as the hierarchy delimiter for the purpose of
resolving paths in relative URLs. IMAP4 permits the use of any
hierarchy delimiter in mailbox names. For this reason, relative
mailbox paths will only work if the mailbox uses "/" as the
hierarchy delimiter. Relative URLs may be used on mailboxes which
use other delimiters, but in that case, the entire mailbox name
MUST be specified in the relative URL or inherited as a whole from
the base URL.

The base URL for a list of mailboxes or messages which was referred
to by an IMAP URL is always the referring IMAP URL itself. The
base URL for a message or message part which was referred to by an
IMAP URL may be more complicated to determine. The program
interpreting the relative URL will have to check the headers of the
MIME entity and any enclosing MIME entities in order to locate the
"Content-Base" and "Content-Location" headers. These headers are
used to determine the base URL as defined in [HTTP]. For example,
if the referring IMAP URL contains a "/;SECTION=1.2" parameter,
then the MIME headers for section 1.2, for section 1, and for the
enclosing message itself SHOULD be checked in that order for
"Content-Base" or "Content-Location" headers.

9. Multinational Considerations

IMAP4 [IMAP4] section 5.1.3 includes a convention for encoding
non-US-ASCII characters in IMAP mailbox names. Because this
convention is private to IMAP, it is necessary to convert IMAP's
encoding to one that can be more easily interpreted by a URL
display program. For this reason, IMAP's modified UTF-7 encoding
for mailboxes MUST be converted to UTF-8 [UTF8]. Since 8-bit
characters are not permitted in URLs, the UTF-8 characters are
encoded as required by the URL specification [BASIC-URL]. Sample
code is included in Appendix A to demonstrate this conversion.

[10](#). **Examples**

     The following examples demonstrate how an IMAP4 client program
     might translate various IMAP4 URLs into a series of IMAP4 commands.
     Commands sent from the client to the server are prefixed with "C:",
     and responses sent from the server to the client are prefixed with
     "S:".

     The URL:

      <imap://minbari.org/gray-council;UIDVALIDITY=385759045/;UID=20>

     Results in the following client commands:

         <connect to minbari.org, port 143>
         C: A001 LOGIN ANONYMOUS sheridan@babylon5.org
         C: A002 SELECT gray-council
         <client verifies the UIDVALIDITY matches>
         C: A003 UID FETCH 20 BODY.PEEK[]

     The URL:

      <imap://michael@minbari.org/users.*;type=list>

     Results in the following client commands:

         <client requests password from user>
         <connect to minbari.org imap server, activate strong encryption>
         C: A001 LOGIN MICHAEL zipper
         C: A002 LIST "" users.*

     The URL:

      <imap://psy.earth/~peter/%E6%97%A5%E6%9C%AC%E8%AA%9E/%E5%8F%B0%E5%8C%97>

     Results in the following client commands:

         <connect to psy.earth, port 143>
         C: A001 LOGIN ANONYMOUS bester@psycop.psy.earth
         C: A002 SELECT ~peter/&ZeVnLIqe-/&U,BTFw-
         <commands the client uses for viewing the contents of a mailbox>

The URL:

  <imap://;AUTH=KERBEROS_V4@minbari.org/gray-council/;uid=20/;section=1.2>

Results in the following client commands:

```
    <connect to minbari.org, port 143>
    C: A001 AUTHENTICATE KERBEROS_V4
    <authentication exchange>
    C: A002 SELECT gray-council
    C: A003 UID FETCH 20 BODY.PEEK[1.2]
```

If the following relative URL is located in that body part:

  <;section=1.4>

This could result in the following client commands:

```
    C: A004 UID FETCH 20 (BODY.PEEK[1.2.MIME]
            BODY.PEEK[1.MIME]
            BODY.PEEK[HEADER.FIELDS (Content-Base Content-Location)])
    <Client looks for Content-Base or Content-Location headers in
     result.  If no such headers, then it does the following>
    C: A005 UID FETCH 20 BODY.PEEK[1.4]
```

The URL:

  <imap://;AUTH=*@minbari.org/gray%20council?SUBJECT%20shadows>

Could result in the following:

```
    <connect to minbari.org, port 143>
    C: A001 CAPABILITY
    S: * CAPABILITY IMAP4rev1 AUTH=GSSAPI
    S: A001 OK
    C: A002 AUTHENTICATE GSSAPI
    <authentication exchange>
    S: A002 OK user lennier authenticated
    C: A003 SELECT "gray council"
    ...
    C: A004 SEARCH SUBJECT shadows
    S: * SEARCH 8 10 13 14 15 16
    S: A004 OK SEARCH completed
    C: A005 FETCH 8,10,13:16 ALL
    ...
```

NOTE: In this final example, the client has implementation dependent
choices. The authentication mechanism could be anything, including

PREAUTH. And the final FETCH command could fetch more or less
information about the messages, depending on what it wishes to display
to the user.

11. Security Considerations

Security considerations discussed in the IMAP specification [IMAP4]
and the URL specification [BASIC-URL] are relevant. Security
considerations related to authenticated URLs are discussed in
section 3 of this document.

Many email clients store the plain text password for later use
after logging into an IMAP server. Such clients MUST NOT use a
stored password in response to an IMAP URL without explicit
permission from the user to supply that password to the specified
host name.

12. ABNF for IMAP URL scheme

This uses ABNF as defined in RFC 822 [IMAIL]. Terminals from the
BNF for IMAP [IMAP4] and URLs [BASIC-URL] are also used. Strings
are not case sensitive and free insertion of linear-white-space is
not permitted.

```
achar             = uchar / "&" / "=" / "~"
                        ; see [BASIC-URL] for "uchar" definition

bchar             = achar / ":" / "@" / "/"

enc_auth_type     = 1*achar
                        ; encoded version of [IMAP-AUTH] "auth_type"

enc_list_mailbox  = 1*bchar
                        ; encoded version of [IMAP4] "list_mailbox"

enc_mailbox       = 1*bchar
                        ; encoded version of [IMAP4] "mailbox"

enc_search        = 1*bchar
                        ; encoded version of search_program below

enc_section       = 1*bchar
                        ; encoded version of section below

enc_user          = 1*achar
                        ; encoded version of [IMAP4] "userid"
```

```
imapurl          = "imap://" iserver "/" [ icommand ]

iauth            = ";AUTH=" ( "*" / enc_auth_type )

icommand         = imailboxlist / ipath / isearch

imailboxlist     = [enc_list_mailbox] ";TYPE=" list_type

ipath            = enc_mailbox [uidvalidity] iuid [isection]

isearch          = enc_mailbox [ "?" enc_search ] [uidvalidity]

isection         = "/;SECTION=" enc_section

iserver          = [iuserauth "@"] hostport
                        ; See [BASIC-URL] for "hostport" definition

iuid             = "/;UID=" nz_number
                        ; See [IMAP4] for "nz_number" definition

iuserauth        = enc_user [iauth] / [enc_user] iauth

list_type        = "LIST" / "LSUB"

search_program   = ["CHARSET" SPACE astring SPACE] 1#search_key
                        ; IMAP4 literals may not be used
                        ; See [IMAP4] for "astring" and "search_key"

section          = section_text / (nz_number *["." nz_number]
                     ["." (section_text / "MIME")])
                        ; See [IMAP4] for "section_text" and "nz_number"

uidvalidity      = ";UIDVALIDITY=" nz_number
                        ; See [IMAP4] for "nz_number" definition
```

13. References

[BASIC-URL] Berners-Lee, Masinter, McCahill, "Uniform Resource
Locators (URL)", RFC 1738, CERN, Xerox Corporation, University of
Minnesota, December 1994.

    <ftp://ds.internic.net/rfc/rfc1738.txt>

[IMAP4] Crispin, M., "Internet Message Access Protocol - Version
4rev1", RFC 2060, University of Washington, December 1996.

    <ftp://ds.internic.net/rfc/rfc2060.txt>

   [IMAP-AUTH] Myers, J., "IMAP4 Authentication Mechanism", RFC 1731,
   Carnegie-Mellon University, December 1994.

       <ftp://ds.internic.net/rfc/rfc1731.txt>

   [HTTP] Fielding, Gettys, Mogul, Frystyk, Berners-Lee, "Hypertext
   Transfer Protocol -- HTTP/1.1", RFC 2068, UC Irvine, DEC, MIT/LCS,
   January 1997.

       <ftp://ds.internic.net/rfc/rfc2068.txt>

   [IMAIL] Crocker, "Standard for the Format of ARPA Internet Text
   Messages", STD 11, RFC 822, University of Delaware, August 1982.

       <ftp://ds.internic.net/rfc/rfc822.txt>

   [KEYWORDS] Bradner, "Key words for use in RFCs to Indicate
   Requirement Levels", RFC 2119, Harvard University, March 1997.

       <ftp://ds.internic.net/rfc/rfc2119.txt>

   [MIME] Freed, N., Borenstein, N., "Multipurpose Internet Mail
   Extensions", RFC 2045, Innosoft, First Virtual, November 1996.

     <ftp://ds.internic.net/rfc/rfc2045.txt>

   [REL-URL] Fielding, "Relative Uniform Resource Locators", RFC 1808,
   UC Irvine, June 1995.

       <ftp://ds.internic.net/rfc/rfc1808.txt>

   [UTF8] Yergeau, F. "UTF-8, a transformation format of Unicode and
   ISO 10646", RFC 2044, Alis Technologies, October 1996.

       <ftp://ds.internic.net/rfc/rfc2044.txt>

14. Author's Address

   Chris Newman
   Innosoft International, Inc.
   1050 East Garvey Ave. South
   West Covina, CA 91790 USA

   Email: chris.newman@innosoft.com

Appendix A. Sample code

Here is sample C source code to convert between URL paths and IMAP
mailbox names, taking into account mapping between IMAP's modified UTF-7
[IMAP4] and hex-encoded UTF-8 which is more appropriate for URLs. This
code has not been rigorously tested nor does it necessarily behave
reasonably with invalid input, but it should serve as a useful example.
This code just converts the mailbox portion of the URL and does not deal
with parameters, query or server components of the URL.

```
#include <stdio.h>
#include <string.h>

/* hexadecimal lookup table */
static char hex[] = "0123456789ABCDEF";

/* URL unsafe printable characters */
static char urlunsafe[] = " \"#%&+:;<=>?@[\\]^`{|}";

/* UTF7 modified base64 alphabet */
static char base64chars[] =
  "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+,";
#define UNDEFINED 64

/* UTF16 definitions */
#define UTF16MASK       0x03FFUL
#define UTF16SHIFT      10
#define UTF16HIGHSTART  0xD800UL
#define UTF16HIGHEND    0xDBFFUL
#define UTF16LOSTART    0xDC00UL
#define UTF16LOEND      0xDFFFUL

/* Convert an IMAP mailbox to a URL path
 *  dst needs to have roughly 4 times the storage space of src
 *    Hex encoding can triple the size of the input
 *    UTF-7 can be slightly denser than UTF-8
 *     (worst case: 8 octets UTF-7 becomes 9 octets UTF-8)
 */
void MailboxToURL(char *dst, char *src)
{
    unsigned char c, i, bitcount;
    unsigned long ucs4, utf16, bitbuf;
    unsigned char base64[256], utf8[6];

    /* initialize modified base64 decoding table */
    memset(base64, UNDEFINED, sizeof (base64));
    for (i = 0; i < sizeof (base64chars); ++i) {
        base64[base64chars[i]] = i;
```

```
    }

    /* loop until end of string */
    while (*src != '\0') {
        c = *src++;
        /* deal with literal characters and &- */
        if (c != '&' || *src == '-') {
            if (c < ' ' || c > '~' || strchr(urlunsafe, c) != NULL) {
                /* hex encode if necessary */
                dst[0] = '%';
                dst[1] = hex[c >> 4];
                dst[2] = hex[c & 0x0f];
                dst += 3;
            } else {
                /* encode literally */
                *dst++ = c;
            }
            /* skip over the '-' if this is an &- sequence */
            if (c == '&') ++src;
        } else {
            /* convert modified UTF-7 -> UTF-16 -> UCS-4 -> UTF-8 -> HEX */
            bitbuf = 0;
            bitcount = 0;
            ucs4 = 0;
            while ((c = base64[(unsigned char) *src]) != UNDEFINED) {
                ++src;
                bitbuf = (bitbuf << 6) | c;
                bitcount += 6;
                /* enough bits for a UTF-16 character? */
                if (bitcount >= 16) {
                    bitcount -= 16;
                    utf16 = (bitcount ? bitbuf >> bitcount : bitbuf) & 0xffff;
                    /* convert UTF16 to UCS4 */
                    if (utf16 >= UTF16HIGHSTART && utf16 <= UTF16HIGHEND) {
                        ucs4 = (utf16 & UTF16MASK) << UTF16SHIFT;
                        continue;
                    } else if (utf16 >= UTF16LOSTART && utf16 <= UTF16LOEND) {
                        ucs4 |= utf16 & UTF16MASK;
                    } else {
                        ucs4 = utf16;
                    }
                    /* convert UTF-16 range of UCS4 to UTF-8 */
                    if (ucs4 <= 0x7fUL) {
                        utf8[0] = ucs4;
                        i = 1;
                    } else if (ucs4 <= 0x7ffUL) {
                        utf8[0] = 0xc0 | (ucs4 >> 6);
                        utf8[1] = 0x80 | (ucs4 & 0x3f);
```

```
                        i = 2;
                    } else if (ucs4 <= 0xffffUL) {
                        utf8[0] = 0xe0 | (ucs4 >> 12);
                        utf8[1] = 0x80 | ((ucs4 >> 6) & 0x3f);
                        utf8[2] = 0x80 | (ucs4 & 0x3f);
                        i = 3;
                    } else {
                        utf8[0] = 0xf0 | (ucs4 >> 18);
                        utf8[1] = 0x80 | ((ucs4 >> 12) & 0x3f);
                        utf8[2] = 0x80 | ((ucs4 >> 6) & 0x3f);
                        utf8[3] = 0x80 | (ucs4 & 0x3f);
                        i = 4;
                    }
                    /* convert utf8 to hex */
                    for (c = 0; c < i; ++c) {
                        dst[0] = '%';
                        dst[1] = hex[utf8[c] >> 4];
                        dst[2] = hex[utf8[c] & 0x0f];
                        dst += 3;
                    }
                }
            }
            /* skip over trailing '-' in modified UTF-7 encoding */
            if (*src == '-') ++src;
        }
    }
    /* terminate destination string */
    *dst = '\0';
}

/* Convert hex coded UTF-8 URL path to modified UTF-7 IMAP mailbox
 *  dst should be about twice the length of src to deal with non-hex coded URLs
 */
void URLtoMailbox(char *dst, char *src)
{
    unsigned int utf8pos, utf8total, i, c, utf7mode, bitstogo, utf16flag;
    unsigned long ucs4, bitbuf;
    unsigned char hextab[256];

    /* initialize hex lookup table */
    memset(hextab, 0, sizeof (hextab));
    for (i = 0; i < sizeof (hex); ++i) {
        hextab[hex[i]] = i;
        if (isupper(hex[i])) hextab[tolower(hex[i])] = i;
    }

    utf7mode = 0;
    utf8total = 0;
```

```
    bitstogo = 0;
    while ((c = *src) != '\0') {
        ++src;
        /* undo hex-encoding */
        if (c == '%' && src[0] != '\0' && src[1] != '\0') {
            c = (hextab[src[0]] << 4) | hextab[src[1]];
            src += 2;
        }
        /* normal character? */
        if (c >= ' ' && c <= '~') {
            /* switch out of UTF-7 mode */
            if (utf7mode) {
                if (bitstogo) {
                    *dst++ = base64chars[(bitbuf << (6 - bitstogo)) & 0x3F];
                }
                *dst++ = '-';
                utf7mode = 0;
            }
            *dst++ = c;
            /* encode '&' as '&-' */
            if (c == '&') {
                *dst++ = '-';
            }
            continue;
        }
        /* switch to UTF-7 mode */
        if (!utf7mode) {
            *dst++ = '&';
            utf7mode = 1;
        }
        /* Encode US-ASCII characters as themselves */
        if (c < 0x80) {
            ucs4 = c;
            utf8total = 1;
        } else if (utf8total) {
            /* save UTF8 bits into UCS4 */
            ucs4 = (ucs4 << 6) | (c & 0x3FUL);
            if (++utf8pos < utf8total) {
                continue;
            }
        } else {
            utf8pos = 1;
            if (c < 0xE0) {
                utf8total = 2;
                ucs4 = c & 0x1F;
            } else if (c < 0xF0) {
                utf8total = 3;
                ucs4 = c & 0x0F;
```

```
            } else {
                /* NOTE: can't convert UTF8 sequences longer than 4 */
                utf8total = 4;
                ucs4 = c & 0x03;
            }
            continue;
        }
        /* loop to split ucs4 into two utf16 chars if necessary */
        utf8total = 0;
        do {
            if (ucs4 > 0xffffUL) {
                bitbuf = (bitbuf << 16) | ((ucs4 >> UTF16SHIFT)
                                           + UTF16HIGHSTART);
                ucs4 = (ucs4 & UTF16MASK) + UTF16LOSTART;
                utf16flag = 1;
            } else {
                bitbuf = (bitbuf << 16) | ucs4;
                utf16flag = 0;
            }
            bitstogo += 16;
            /* spew out base64 */
            while (bitstogo >= 6) {
                bitstogo -= 6;
                *dst++ = base64chars[(bitstogo ? (bitbuf >> bitstogo) : bitbuf)
                             & 0x3F];
            }
        } while (utf16flag);
    }
    /* if in UTF-7 mode, finish in ASCII */
    if (utf7mode) {
        if (bitstogo) {
            *dst++ = base64chars[(bitbuf << (6 - bitstogo)) & 0x3F];
        }
        *dst++ = '-';
    }
    /* tie off string */
    *dst = '\0';
}
```