print | close

Base64 Is Built In to IBM i

System iNetwork Programming Tips Newsletter

Scott Klement

Scott Klement

Thu, 2009-07-09 (All day)

Base64, a popular method of encoding data, allows 8-bit binary values to be used in a medium that supports only simple text characters. For example, the SMTP protocol used to exchange data over the Internet supports only 7-bit ASCII characters. In order to send data that's not legal 7-bit ASCII text (e.g., pictures, office documents, video files, etc.) over SMTP, it must be Base64 encoded.

Although I released open-source Base64 routines a while back, I recently received an email message from an RPG programmer who discovered that there are Base64 routines included in IBM i! This article tells you how to use those routines in your own RPG programs.

Common Uses of Base64

- Encoding user names and passwords in the HTTP protocol
- Allowing international characters in 7-bit ASCII text fields
- Attaching binary documents to email messages
- Including binary documents inside an XML document
- Encoding binary digital keys or certificates for use in encryption
- Embedding images in a CSS stylesheet
- Passing profile tokens to web applications
- In general, anytime you need to include the full range of text characters (including control characters and international characters) in a plain-text field
- In general, anytime you need to include binary data in a plain-text field

Of course, if you're using prewritten software that does any of the preceding tasks, it's already taking care of the Base64 encoding and decoding for you, under the covers. However, if you want to write your own code to do any of the preceding tasks, you'll need to find routines for Base64 encoding or decoding. In that case, having something included with IBM i would be convenient, wouldn't it?

You'll notice that these uses fall into two categories. Sometimes you use it to encode text characters (in blue, above) or to encode binary data (in red, above.) On i, this is an important distinction. Since Base64 always decodes back to the same *binary* byte values that were encoded, you must make sure that any text is translated to ASCII prior to encoding it. By contrast, binary data, such as an image or Excel document, isn't made up of text characters and therefore should never be translated from EBCDIC to ASCII (or vice versa), since it's not made up of EBCDIC characters to begin with. Instead you should be careful to ensure that it's never translated at all.

Encoding Not a Cipher

Although Base64 is used in conjunction with encryption and password exchange, I want to clarify that by itself, it's not a form of cryptography. For example, if I have a secret password and Base64 encode it, the result might look like this:

```
WW91J3ZlIGZvdW5kIG15IHNlY3JldCEK
```

It's tempting to think that the preceding string is encrypted and protected from prying eyes. After all, it looks like a string of random letters and numbers, doesn't it? However, although it's obfuscated, it's not encrypted. Anyone can easily decode that string and see what it says. For example, you could simply paste the string into a free website and the site would show you the result.

Since there's no passphrase or digital key involved, it's very easy to see what's behind a Base64 string. So don't make the mistake of thinking it's encrypted! However, it's used quite often with cryptography. The best and most secure ciphers output encrypted data in a binary format incompatible with plain-text format. If you need to send that encrypted data over a network, Base64 is a great tool to enable you to send the encrypted data from one place to another.

Apache Portable Runtime and QAXIS10HT

The Base64 routines are included as part of the Apache Portable Runtime (APR), which is a set of routines provided by Apache to make it easier for software to work on multiple platforms. For example, if I write a C program for Windows that uses the APR routines, I can move that C program to IBM i and recompile it, and theoretically my code will work exactly as it did on Windows.

On i, the APR routines are included with the Apache AXIS for C++ toolkit.

The Integrated Web Services (IWS) client software for IBM i is based on the Apache AXIS for C++ toolkit. This IWS toolkit is free and included with IBM i. It requires IBM i 5.4 or higher, and on 5.4 and 6.1, you need to install a PTF to enable it.

To see if the support is already included on your system, type the following command:

```
DSPSRVPGM SRVPGM(QSYSDIR/QAXIS10HT) DETAIL(*PROCEXP)
```

Look through the list of procedure exports, and see if you can find one named apr_Base64_decode. If that export is found, then your system has the Apache Axis for C++ toolkit installed, and you can use the APIs in it.

Unfortunately, I don't know the PTF number that enabled this support, and the IBM documentation is unclear on that point. However, it does appear that the service program named QAXIS10HT is part of the 57xx-SS1 option 3 (Extended Base Directory Support), which is a required component of IBM i. So starting with the next release (the one that comes after 6.1), this service program should always be on the system, and therefore you'll always have a Base64 encoder and decoder available.

Routines Available

Here are the routines available for Base64 encoding/decoding in the APR utilities:

int apr_base64_encode_len (int len) int apr_base64_encode (char *coded_dst, const char *plain_src, int len_plain_src) int apr_base64_encode_binary (char *coded_dst, const unsigned char *plain_src, int len_plain_src) int apr_base64_decode_len (const char *coded_src) int apr_base64_decode (char *plain_dst, const char *coded_src) int apr_base64_decode_binary (unsigned char *plain_dst, const char *coded_src)

- **apr_base64_encode_len**: given the length of the plain-text (unencoded) data, calculates how long the encoded data would be.
- **apr_base64_encode**: first translates data to ASCII then encodes it. Useful for encoding text data.
- **apr_base64_encode_binary**: encodes data. Same as apr_base64_encode, except that data is not translated to ASCII first. Useful for encoding binary data.
- **apr_base64_decode_len**: Given a Base64-encoded string, calculates the length of the plain-text string after it has been decoded.

- **apr_base64_decode**: Takes a Base64-encoded string, decodes it, and converts the result to EBCDIC. Useful for decoding text data.
- **apr_base64_decode_binary**: Takes a Base64-encoded string and decodes it. Useful for decoding binary data.

I've written a copy book (included in the code download at the end of this article) that contains the prototypes for these six routines. I've also included sample code that demonstrates how to write an email client that Base64 encodes a PDF document before adding it to the email.

Example: Email with Base64 Encoded Attachment

Because email attachments are the most common use for Base64, it make sense to use that as my example. I've written a program that uses the QtmmSendMail() API to send an email (though you could easily modify it to use a different means). It provides a message to the user and then attaches a PDF file from the IFS as a Base64-encoded document.

One important aspect of Base64 is that it always encodes three plain-text bytes and produces four encoded bytes. The only exception to this is when you reach the end of the file and therefore have less than three bytes of input. In that case, the output is still four bytes, but the input may be shorter. Base64 assumes that anytime it encounters short input, it has reached the end of the file you're encoding.

It might be tempting to think that I'd have to read the entire file into a string in memory then run it through the encoder in one fell swoop, but this isn't really necessary. The four/three rule described in the preceding paragraph means that I can read the file in chunks and encode it in chunks, provided that the chunk size is a multiple of three bytes. It won't think it's at the end of the file until I read a chunk that's not a multiple of three bytes, and that shouldn't occur until I hit the end of the PDF file that I'm encoding.

With all that in mind, the encoded email message should look something like this:

```
From: Scott Klement
To: Faithful Reader
Date: Wed, 09 Jul 2009 05:36:40 -0500
Subject: Testing sending a PDF file
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="--=_ScottsNiftyBoundary"

Your mail reader doesn't support MIME!

----=_ScottsNiftyBoundary
Content-type: text/plain

Hi there,

The purpose of this message is to send you the attached
file. You can open it in Adobe Reader or any other
program that understands PDF files.

Good luck!
----=_ScottsNiftyBoundary
Content-Type: application/pdf; name="Employment Application.pdf"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="Employment Application.pdf"
```

```
JVBERi0xLjQKJeLjz9MKNiAwIG9iago8PCAKICAgL1R5cGUgL1hPYmplY3QKICAgL1N1YnR5
cGUgL0ltYWdlCiAgIC9CaXRzUGVyQ29tcG9uZW50IDgKICAgL1dpZHRoIDc4NgogICAvSGVp
Z2h0IDQ0NwogICAvQ29sb3JTcGFjZSAvRGV2aWNlUkdCCiAgIC9GaWx0ZXIgL0RDVERlY29k
.
.
more Base64-encoded data
.
.
cmFpbGVyCjw8IAogICAvUm9vdCAxIDAgUgogICAvSW5mbyA1IDAgUgogICAvU2l6ZSAxNQo+
PgpzdGFydHhyZWYKNTYxMTgKJSVFT0YK
```

```
----=_ScottsNiftyBoundary--
```

MIME email messages are split up into "parts," where each part represents either a part of the message or an attachment to the message. It starts with a series of keywords and their values. From, To, Date, and Subject are all keywords. The MIME-version and Content-Type keywords are especially important because they tell the email software how to interpret the rest of the message. The fact that it shows MIME-version means that it is a MIME message and should be processed like one. The fact that the Content-Type shows multipart/mixed means that this message will have many parts. The boundary variable is set to ScottsNiftyBoundary, which will be used to split up the rest of the message into parts--an email client will search for the boundary string and will use that to separate the various pieces of the email message. Each part starts out with keywords, one per line, followed by a blank line that tells the system you've reached the end of the headers. The remainder of the message part is the body and contains the data for that part of the message.

The following is an excerpt that shows how this file data would be read from one file (the PDF file to be encoded), Base64 encoded, and attached to the email. This only shows the attachment portion of the message.

```
body =
 '--' + boundary + CRLF
+'Content-Type: ' + mimeType + '; '
+    'name="' + attname + '"' + CRLF
+'Content-Transfer-Encoding: base64' + CRLF
+'Content-Disposition: attachment; '
+    'filename="' + attname + '"' + CRLF
+ CRLF;
callp write(fd: %addr(body)+2: %len(body));

att = open( attFile: O_RDONLY );
if (att = -1);
   ReportError();
endif;

dow '1';
   len = read(att: %addr(data): %size(data));
   if (len              enclen = apr_base64_encode_binary( encdata
                                  : data
                                  : len ) - 1;
   %subst(encdata:enclen+1) = CRLF;
   callp write(fd: %addr(encdata): enclen+2);
enddo;

callp close(att);
```

You'll notice that I'm calling the apr_base64_encode_binary() and passing the data that I read from the IFS file, as well as the length of the data from the IFS file. The APR routine dutifully Base64 encodes my data and returns it in the encdata field. It also returns the size of the encdata field, so I know how much to write to disk. Interestingly, the length returned always seems to be one byte too high. I assume this is because it's a C routine, and C always adds a x'00' (null-terminator) to the end of character strings. That extra character would explain the length always being one number too high. You'll notice in the preceding code that I subtract one from the length and then use that length to insert a CRLF at the end of the record before finally writing it to disk.

Code Download

You can download the sample code for this article (including the APR_B64_H copy book and the sample program to attach a PDF to an email).

Previous Articles

- SHA1, MD5, and Base64 Without APIs
- E-mail Message Bodies
- Send E-mail Messages with SMTP

Source URL: http://iprodeveloper.com/application-development/base64-built-ibm-i