Base64 MIME Encoding

by Damon Matthews

Although email is a good means of communicating with people all over the world, it does suffer from one distinct disadvantage. It is purely text based and cannot transfer binary data unaided. This disadvantage, together with the obvious need to transfer binary data, led to the creation of MIME, or Multipurpose Internet Mail Extensions.

MIME is a method of allowing the attachment of binary data to a standard email by first converting that data into a textual format, which can then be restored to its original format upon receipt.

MIME Data Types

According to the MIME RFC (Request For Comment), there are seven basic types of data that can be transferred using this method. These are: text, multipart (for those containing multiple pieces of independent data), message (the message body is itself a full email message including all headers and any MIME attachments), image, audio, video and application.

Any form of data can usually be placed in one of these categories as a subtype. For instance, a JPEG image is of type image/jpeg and a wav file is of type audio/wav. Anything that cannot be easily placed in any of these types is given a type beginning with an x.

All standard types (that is, those not beginning with an x) are agreed standards which form part of the MIME standard. So if you were to invent a new image type and wanted it to be MIME encoded it would come under x until such time as it was integrated into the standard MIME subtypes. So, for instance, you could call it image/x-myimage (rather than image/myimage).

MIME is fairly extensive, being contained in a multi part RFC. The first part of this is RFC 1521, which covers the basic outline of MIME and the techniques used to convert binary data into a textual format and back again. This RFC is nearly 70 pages in length and could be considered heavy reading. It's really only necessary for those needing to fully implement MIME (or maybe those with an interest in internet protocols and/or nothing better to read!). There are also additional RFCs covering specialised forms of MIME encoding such as that used with EDI (Electronic Data Interchange).

What I am going to cover here is one of the methods for carrying out the actual encoding and decoding of binary data for attachment to an email: Base64.

There are four main methods of carrying out encoding and decoding. These are: Quoted Printable, UU-Coding, XX-Coding and Base64 Coding. Of these, Base64 is probably the best, as the others have some disadvantages. Quoted Printable is limited and is not ideal for transferring a binary file: it is more suited to converting the occasional non-standard character that cannot be sent via email (such as accented characters or special characters like the English pound sign).

UU and XX encoding are better, allowing the transfer of binary files, but have the disadvantage of not using a universally standard set of translation characters. If either of these pass through a system using, for instance, EBCDIC, there is a chance they can be altered and therefore corrupted (this may only be a small chance, but it might happen, as outlined in the MIME RFC). Base64 has the advantage of being identically represented in ASCII and all forms of EBCDIC.

Base64

Base64 works in the following way. A binary input stream is taken 3 binary characters at a time, giving 24 bits of information. This is converted into 4 groups of 6 bits, which are used as an index into an array of 64 characters (the Base64 alphabet). So, for each 3 binary characters read in, 4 ASCII (Base64) characters are produced. The output stream is then cut at every 76 output characters (ie has a line length of 76). This means that each line of a Base64 encoded file represents 76/4*3 or 57 binary characters.

Now, the number of characters in the finally encoded file must be exactly divisible by 4. This means that the number of binary characters in the input file would need to be divisible by 3. Of course it is unrealistic to expect this. So, there are three possible cases that can occur at the end of a file. There will be 1, 2 or 3 characters. If there are 3 characters, then all is fine. If there are 1 or 2, then the encoded output must be padded to make a total of 4 characters.

The padding is done like this. If there are 2 characters left, these two characters are used to create 3 Base64 characters and a single = character is added onto the end. If there is 1 character left, the single character is used to create 2 Base 64 characters and 2 = characters are added on the end.

Base64 uses the following character set: the upper case alphabetic characters, the lower case alphabetic characters, the numbers 0 to 9 and the characters + and /. The padding character = is used in addition and is not counted as one of the 64 characters in the Base64 alphabet (and can therefore appear only at the end of the encoded file: if it occurs at any other place in the encoded file then something is wrong).

Implementation

So how would one go about implementing MIME encoding and decoding in Delphi? There are a couple of options available. You could of course look for a component to do it, and there are a few available. Secondly, if you have

```
function TForm1.encode: boolean;
var
  encodedString: String;
  setof3ASCII: Array[1..3] of byte;
  counter: Integer;
  ord1,ord2,ord3: Integer;
  file2enc: TFileStream;
  read57: Array[1..57] of byte;
  strlength: LongInt;
  loops: Integer;
begin
  file2enc := TFileStream.Create(infilename,fmOpenRead);
  encodedfile := TStringList.Create;
  strlength := file2enc.Read(read57,57);
  while strlength > 0 do begin
    // beginning of main loop
    encodedString := '';
    loops := strlength div 3;
    for counter := 0 to loops -1 do begin
      setof3ASCII[1] := read57[counter*3 +1];
      setof3ASCII[2] := read57[counter*3 +2];
      setof3ASCII[3] := read57[counter*3 +3];
      ord1 := ord(setof3ASCII[1]);
      ord2 := ord(setof3ASCII[2]);
      ord3 := ord(setof3ASCII[3]);
      encodedString := encodedString +
        base64Alphabet[(ord1 div 4)+1];
      encodedString := encodedString +
        base64Alphabet[(ord1 mod 4)*16 + (ord2 div 16)+1];
      encodedString := encodedString +
        base64Alphabet[(ord2 mod 16)*4 +(ord3 div 64)+1];
      encodedString := encodedString +
        base64Alphabet[ord3 mod 64+1];
    end;
    // two characters left over at end
    if( strlength mod 3 = 2) then begin
      setof3ASCII[1] := read57[strlength-1];
      setof3ASCII[2] := read57[strlength];
      ord1 := ord(setof3ASCII[1]);
      ord2 := ord(setof3ASCII[2]);
      EncodedString := encodedString +
        base64Alphabet[(ord1 div 4)+1];
      EncodedString := encodedString +
        base64Alphabet[(ord1 mod 4)*16 +(ord2 div 16)+1];
      EncodedString := encodedString +
        base64Alphabet[(ord2 mod 16)*4 +1];
      EncodedString := encodedString + '=';
    end;
    // one character left over at end
    if( strlength mod 3 = 1) then begin
      setof3ASCII[1] := read57[strlength];
      ord1 := ord(setof3ASCII[1]);
      EncodedString := encodedString +
        base64Alphabet[(ord1 div 4)+1];
      EncodedString := encodedString +
        base64Alphabet[(ord1 mod 4)*16 +1];
      EncodedString := encodedString + '=';
      EncodedString := encodedString+'=';
    end;
    encodedfile.Add(encodedString);
    strlength := file2enc.Read(read57,57);
  end; // end of main loop
  encodedfile.SaveToFile(outfilename);
  file2enc.Free;
  Result := true;
end;
```

➤ *Listing 1*

Delphi 4 Professional there is a relevant component. The third option is of course to write it yourself.

So, to do the job ourselves, we need functions to encode and decode a file. The first thing is to declare the Base64 Alphabet (or character set if you prefer). It can easily be declared as a `TString` and populated as follows:

```
Base64Alphabet :=
  'ABCDEFGHIJKLMNOPQRSTUVWXYZ'+
  'Abcdefghijklmnopqrstuvwxyz'+
  '0123456789+/';
```

To encode a file, first we open the file (using a `TFileStream`), then create an array of 76 bytes to hold the converted characters. Next we read in 3 (binary) characters. If we succeeded in getting 3 characters, carry out the conversion and add them to the array. If we only got 2 characters, convert these into 3 characters, add a single = and add them to the array. If we only got 1 character, convert this into 2 characters, add two = characters and add them to the array. Then, if the array holds 76 characters, add it to a `TStringList` (which is ideal for holding the converted file). If we're not at the end of the file, we then attempt to read another 3 characters. Listing 1 shows the code.

To decode a file, first we read the file into a `TStringList`. Then we open a `TFileStream` (to write the converted MIME file out to). Next we read a line from the `TStringList`. If the line is 76 characters long, we read 4 characters and convert them into 3 binary characters, then write these out to the `TFileStream`. If we've not yet converted all 76 characters we read another 4 characters. If the line is not 76 characters long, we read 4 characters and convert them into 3 binary characters, then write these out to the `TFileStream`, continuing until you reach the final 4 characters. Then, with the final 4 characters: if there is one = at the end take the other 3

➤ *Listing 2*

```
function TForm1.decode: boolean;
var
  stringToDecode: String;
  setof4ASCII: Array[1..4] of char;
  setof57: Array[1..57] of char;
  maincounter, counter: Integer;
  ord1,ord2,ord3, ord4: Integer;
  use3rd, use4th: boolean;
  outf: TFileStream;
  total: Integer;
  loops: Integer;
begin
  encodedfile.LoadFromFile(infilename);
  outf := TFileStream.Create(outfilename,fmCreate);
  for maincounter := 0 to encodedfile.count - 1 do begin
    // beginning of main loop
    stringToDecode := encodedfile.Strings[maincounter];
    total := 0;
    loops := strlen(PChar(stringToDecode)) div 4;
    for counter := 0 to loops-1 do begin
      use3rd := true;
      use4th := true;
      setof4ASCII[1] := stringTodecode[counter*4 +1];
      setof4ASCII[2] := stringTodecode[counter*4 +2];
      setof4ASCII[3] := stringTodecode[counter*4 +3];
      setof4ASCII[4] := stringTodecode[counter*4 +4];
      ord1 := pos(setof4ASCII[1],base64Alphabet)-1;
      ord2 := pos(setof4ASCII[2],base64Alphabet)-1;
      if(setof4ASCII[3] = '=') then begin
        ord3 := 0;
        use3rd := false;
      end else
        ord3 := pos(setof4ASCII[3],base64Alphabet)-1;
      if(setof4ASCII[4] = '=') then begin
        ord4 := 0;
        use4th := false;
      end else
        ord4 := pos(setof4ASCII[4],base64Alphabet)-1;
      setof57[counter*3+1] := chr(ord1*4+(ord2 div 16));
      if use3rd then begin
        setof57[counter*3+2] :=
          chr((ord2 mod 16)*16 + (ord3 div 4) );
        inc(total);
      end;
      if use4th then begin
        setof57[counter*3+3] := chr((ord3 mod 4)*64 +ord4);
        inc(total);
      end;
      inc(total);
    end;
    if total > 0 then
      for counter:=1 to total do
        outf.Write(setof57[counter],1);
  end; // end of main loop
  outf.Free;
  Result:=true;
end;
```

characters and convert them into 2 binary characters, if there are two = characters take the two MIME characters and convert them into 1 binary character (note this should only occur on the final line of the `TStringList`: if it doesn't, something is wrong with the encoded file). We continue reading and processing lines until the whole `TStringList` is dealt with. Listing 2 shows the code.

Most of this, as can be seen from the listings, is relatively simple. The only somewhat complicated part is the actual conversion from three 8-bit characters to four 6-bit characters and four 6-bit characters to three 8-bit characters (and of course the conversion when there are a fewer number of 8-bit or 6-bit characters).

There are various techniques that can be used for this: the one I have chosen is to perform a calculation on the values of the characters (using the `ord` function) and using this as an index into the array holding the Base64 alphabet. The actual calculations can be seen in Listings 1 and 2. If you look at the source for other MIME encoding components, you will see this and other methods used to carry out the conversion.

There is of course much more to MIME than this. I've just covered Base64 encoding. To fully implement MIME, you would need to be able to handle the other forms of encoding, as well as the various wrappers placed around the encoded files. This includes the MIME headers, separators and the name of encoded file. Looking at the headers of any email with an attachment will show these, and they are all fully covered in RFC 1521. If you wish to read the RFC, it can be found at http://sunsite.doc.ic.ac.uk/rfc /Rfc1521.txt or if you prefer a postscript file http://src.doc.ic.ac.uk/rfc/Rfc1521.ps

Damon Matthews works at the UK Inprise/Borland User Group and can be reached by email as damon@richplum.co.uk