Base-64 Encoding

Multipurpose Internet Mail Extensions (MIME) is the official format for internet emails. Emails are transmitted to recipients via the **SMTP** (Simple Mail Transfer Protocol), which only supports the transfer of **7-bit ASCII characters**. To handle binary data in email attachments, the MIME standard provides several encoding methods, one of the most common being **Base-64 encoding**.

Below is an example of an email with an attachment, showing the source data that email programs interpret:

```
Content-type: multipart/mixed; boundary="frontier"
MIME-version: 1.0
--frontier
Content-type: text/plain
This is the body of the message.
--frontier
Content-type: application/octet-stream
Content-transfer-encoding: base64
gajwO4+n2Fy4FV3V7zD9awd7uG8/TITP/vIocxXnnf/5mjgQjcipBUL1b3uyLwAVtBLOP4nV
AhSzlZnyLAF8na0n7g6OSeej7EjlF/aglS6ghfju FgRr+OX==
--frontier--
```

The second and third lines from the end show **Base-64 encoded data**.

What is Base-64 Encoding?

Base-64 encoding (or more generally, data representation) is based on a set of 64 symbols. It's essentially like converting data into a 64-base numbering system. The encoding works by grouping the data into **6-bit chunks**.

The Base-64 encoding scheme uses the following symbol set:

- **0..25** → 'A'..'Z'
- **26..51** → 'a'..'z'
- **52..61** → '0'..'9'
- **62** → '+'
- **63** → '/'

This gives us 64 unique symbols to represent binary data. The encoding is done so that every **3 bytes** of input data is converted into **4 characters** of Base-64 output.

Example of Base-64 Encoding

Let's take a simple binary string and encode it in Base-64:

Input: `001100110011`

We split it into two 6-bit groups:

```
001100 , 110011
```

These groups correspond to decimal values **12** and **51**. Using the Base-64 table:

- 12 → 'M'
- 51 → 'z'

Thus, the Base-64 encoding of `001100110011` is **Mz**.

Padding and Handling Non-Multiples of 3 Bytes

Base-64 encoding operates on blocks of 3 bytes (24 bits). If the input data isn't divisible by 3, padding is added to complete the block. This padding is represented by the **'='** symbol. Here are two examples:

Example 1

Encoding the byte `00000001`

1. We first extend it to 3 bytes by adding two zero bytes: `00000001 00000000 00000000`.

2. Split this into 6-bit groups: `000000 010000 000000 000000`.

3. The result is AQ==.

Example 2:

Encoding the bytes `00000010 00000001`

1. Extend to 3 bytes: `00000010 00000001 00000000`.

2. Split into 6-bit groups: `000000 100000 000100 000000`.

3. The result is AgE=.

Base-64 Decoding

Decoding Base-64 involves reversing the process, converting each Base-64 character back into its 6-bit binary equivalent. The binary groups are then combined to form the original data. The number of **'='** symbols at the end of the encoded data tells us whether the last byte is incomplete, helping determine whether to interpret the last 6 or 12 bits of the encoded string.

Special Considerations

- Base-64 encoded data can contain **line break characters** to help manage long strings. These line breaks (or any other non-Base-64 characters) should be ignored during decoding.
- One major advantage of Base-64 encoding is that it safely encodes binary data into a **text-friendly format**, making it ideal for transmission over protocols that only support text (like email and HTTP headers).

Real-World Application of Base-64

In everyday internet use, Base-64 encoding is used in various places, such as:

- Email attachments (as shown in the MIME example).
- Embedding image data in HTML or CSS files.
- Encoding sensitive data in URL query parameters to ensure safe transmission.

For instance, in the email example above, the binary content of the attachment is encoded into Base-64 so it can be transmitted through the SMTP protocol, which only supports 7-bit ASCII characters.