Introduction à MIME: Multipurpose Internet Mail Extension

Lorsque les messages électroniques ont été créés, il y a 20 ans, le simple fait d'envoyer quelques caractères était déjà une belle réussite. Aujourd'hui, on veut envoyer des messages, des images, du son, de la vidéo, MIME permet de décrire ces nouveaux contenus.

Les échanges de messages électroniques ont débuté, il y a vingt ans. A cette époque (préhistorique sur l'échelle électronique), on a forcément fait avec les technologies du moment, les messages ne pouvaient contenir que du texte au format ASCII, pas de ligne de plus de mille caractères et une taille de message limitée (c'était les SMS du moment). En juin 1992, un nouveau standard a été approuvé. Ce standard est MIME: Multipurpose Internet Mail Extension. Il permet d'étendre les fonctionnalités des anciens protocoles, par l'ajout de nouveaux champs (voir articles précédents sur SMTP, POP et IMAP). On a ainsi la possibilité d'envoyer des messages plus longs, des fichiers attachés, des images, des vidéos ou du texte enrichi (caractères accentués, html, …)…

En bref, MIME permet la création de messages contenants:

- Plusieurs objets dans un seul message
- Des caractères autres que ceux définis par la norme ASCII
- Des messages de tailles quasi illimitées
- Des fichiers binaires ou spécifiques à une application
- Des messages multimédia (audio, vidéo, image)
- Du texte enrichi (plusieurs polices de caractères, couleur…)

Exemple

From: "Nicolas" <nicolas@salemioche.com>
To: <nicolas@salemioche.servebeer.com>
Subject: test d'encodage MIME
MIME-Version: 1.0
Content-Type: multipart/mixed;
boundary=-=_NextPart"

This is a multi-part message in MIME format.

---=_NextPart
Content-Type: text/plain;
charset="us-ascii"
Content-Transfer-Encoding: 7bit

ceci est un message au format texte
il y a un attachement.

---=_NextPart
Content-Type: application/octet-stream;
name="Image1.png"
Content-Transfer-Encoding: base64
Content-Disposition: attachment;
filename="Image1.png"

iVBORw0KGgoAAAANSUhEUgAAAAEAAAABCAIAAACQd1PeAAAAL3RFWHRDcmVhdGlvbiBUaW1lAHZl
bi4gMTcgamFudi4gMjAwMyAxMjozMDowNyArMDEwMDJfaQ8AAAAHdElNRQfTARELHzhkt
oSKAAAA
CXBIWXMAAAsSAAALEgHS3X78AAAABGdBTUEAALGPC/xhBQAAAAxJREFUeNpj+P//PwA
F/gL+MxKV
FAAAAABJRU5ErkJggg==

---=_NextPart--
.

MIME est une extension des protocoles de messagerie électronique, le document ci-dessus est un email que je me suis envoyé. Il contient donc les champs standards et courants tels que From:, To: ou Subject:. Il contient aussi de nouveaux champs qui font partie des extensions telles que MIME-Version, Content-Type.

Un message MIME peut contenir plusieurs objets, appelé *body part*. Chaque *body part* est séparé par une ligne incluant la chaîne de caractères boundary (ici -=_*NextPart*). Chaque *body part* est un message compatible avec le standard défini par la RFC 822. Cela veut dire que chaque partie est formé comme un message avec une entête et un corps. L'entête étant constituée de plusieurs champs et de leur valeur. Le corps (ou body) est séparé de l'entête d'une ligne blanche. De plus MIME est récursif et un *body part* peut être de type *multipart* et contenir plusieurs *body part*.

Syntaxe

La standard MIME est une extension des protocoles d'échanges de messages électroniques, pour permettre aux applications de reconnaître un email MIME, le champ *MIME-Version* est obligatoire dans l'entête. Aujourd'hui la seule version possible est 1.0.

On doit ensuite définir le type de message MIME qui suit. C'est le *Content-Type*, cette ligne est toujours de la forme:

Content-Type: type / sous-type ; [paramètres]

Les paramètres dépendent du type et du sous-type du message. Dans l'exemple ci-dessus vous retrouvez plusieurs exemples avec leur paramètre associé.
Les champs sont ensuite suivis d'une ligne blanche comme pour un email standard, mais le corps utile du message ne débute ici qu'avec la première ligne *boundary*.

Types

Voyons un peu la liste des types disponibles, qui permettent de faire de jolis emails.

- **Application** : c'est le type *polyvalent* (ou poubelle selon votre point de vue), on l'utilise pour tous les cas non définis ailleurs.
    - **Application/Octet-Stream**: indique des données binaires de type quelconque. On trouvera la plupart des attachements que l'on utilise quotidiennement comme les fichiers compressés, les documents… On pourra utiliser le paramètre *Name* pour nommer l'objet.
    - **Application/PostScript**

- **Audio :** le seul sous-type disponible est basic (son mono, échantillonné à 8kHz) qui présente peut d'intérêt aujourd'hui, on utilisera donc généralement Application/Octet-Stream.

- **Vidéo :** avec le sous-type Mpeg (Video/Mpeg), on définit un objet vidéo encodé au standard MPEG.

- **Image**
    - **Image/Jpeg :** image au format Jpeg
    - **Image/Gif :** image au format Gif

    Seul ces deux formats sont définis, voila pourquoi on utilisera aussi beaucoup le type Octet-Stream pour pallier ce manque.

- **Message**
    - **Message/Rfc822 :** permet d'encapsuler un email
    - les autres sous-types sont peu usités

- **Multipart :** C'est là que MIME prend toute son ampleur, avec la possibilité de composer son message avec plusieurs objets. C'est pour cette raison que MIME est souvent associé à l'usage des attachements (ou fichiers joints). C'est en fait dans ce cas qu'on le retrouve le plus souvent. Pour définir un type Multipart, il est obligatoire de spécifier le paramètre boundary. Ce paramètre permet de définir le début et la fin de chaque objet. Le début d'un objet est précédé d'un ligne comprenant un double tiret (--) et la chaîne définie comme boundary. A la fin du dernier objet on ajoute le double tirets a cet ensemble (--boudary--). On prendra soit de s'assurer que ces chaînes de caractères ne peuvent pas apparaître dans le corps d'un objet!!! De même un objet pouvant lui aussi être de type Multipart, il faudra alors définir un deuxième boundary différent du premier.

    - **Multipart/Mixed :** le grand classique qui indique plusieurs *body parts* (objets) à présenter en série. On à donc une ligne du type :
      Content-Type: Multipart/Mixed; boundary="__delimiteur___"

    - **Multipart/Alternative :** le format est le même que pour mixed, mais chaque objet représente le même contenu de façon différente. Le client pourra alors choisir celui qui correspond le plus à ces goûts ou ces possibilités d'affichage. On pourra ainsi avoir un même texte présenté par exemple en HTML ou en texte pure.

  - **Multipart/Parallel :** cette fois le contenu des objets sera exécuté en parallèle, on pourra ainsi avoir une vidéo et du son.

- **Text :** pour ces objets on pourra utiliser le paramètre *Charset* pour définir le jeu de caractères utilisé (US-ASCII, ISO-8859-1, …)

  - **Text/Plain :** du texte, rien que du texte, sans fantaisie.

  - **Text/Richtext :** pour un peu plus de gaîté et de couleur, on peut utiliser un sous ensemble de SGML pour écrire son texte.

  - **Text/Html :** comme on peut l'imaginer, cela permet d'envoyer du texte en utilisant le formant html.

- **X-** : suivi de ce que l'on veut. On peut définir ses propres types privés et ainsi étendre les fonctionnalités de MIME.

C'est un peu rébarbatif et longuet, je n'ai donc fait figurer que les types principaux. Pour obtenir une liste complète et exhaustive des types, consultez les RFCs.

Encodage

Pour mémoire, on se souvient que SMTP ne supporte à l'origine que des lignes courtes encodées avec un jeu de caractères sur 7 bits. Or avec la multitude de types décrite ci-dessus, cela ne va pas être facile. Pour pallier cette limitation, chaque objet peut être encodé. On utilisera pour cela le champ *Content-Transfer-Encoding*. Les valeurs possible pour ce champs sont: BASE64, QUOTED-PRINTABLE, 8BIT, 7BIT, BINARY, x-EncodingName (MIME est un standard ouvert, on pourra donc définir ses propres encodeurs).

7bit est utilisé lorsque le message est constitué exclusivement par des caractères us-ascii, il n'y a donc pas d'encodage.

Quoted-printable est très similaire, il permet d'encoder un message principalement constitué de caractères us-ascii, les autres sont *escapés* à l'aide du signe égal (=) et de 2 chiffres hexadécimaux. Ce format offre l'avantage de resté lisible même si les lignes doivent être découpées en segment de 76 caractères.

Pour 8bit et binary, il n'y a pas d'encodage, il faut donc que les relais SMTP supportent cette possibilité. Contrairement à binary, 8bit indique que les lignes du messages sont courtes (c'est-à-dire moins de 76 caractères).

Base64 permet d'encoder n'importe quel objet avec un jeu de caractères 7bits. En théorie le document résultat doit être un suite de ligne de longueur maximum 76 caractères.

Base64

Voyons quand même un peu de code avec les spécificités de l'encodage base64. L'encodage base64 est très simple, on prend des données encodées sur 8bits et on les modifie pour qu'il n'y ait plus que 7 bits utiles. Cela permet de transférer des informations codées en 8 bits sur des équipements ne supportant que le 7bits (comme le protocole SMTP).

Pour cela on prend les données par paquet de trois octets. On a donc 24bits. On va découper ces 24bits en 4 paquets égaux de 6 bits que l'on va ensuite translater dans une table de 7bits. On a donc 3 octets encodés dans 4 à l'arrivée (ce n'est pas très efficace pour une gestion tendue de la bande passante). Chaque paquet de 6 bits est donc un entier compris entre 0 et 63 (d'où base64). Les caractères sont translatés pour utiliser les caractères suivants:

- De 0 à 25 : A à Z
- De 26 à 51 : a à z
- De 52 à 61 : 0 à 9
- 62 : +
- 63 : /

Le résultat est un message encodé qui contient un nombre de caractères multiple de 4. Si il n'y a pas suffisamment de données, on complète avec le caractère égal (=). On utilise donc au total 65 caractères, il faut donc que le système supporte l'encodage en 7bits.

Pour le décodage, c'est très simple, on prend un paquet de 4 octets que l'on va encoder pour retrouver les 3 octets d'origine.

Voyons un exemple d'encodage et de décodage de données au format base64. Les codes qui suivent sont écrit en C et permettent l'encodage/décodage d'un fichier ou des données de l'entrée standard. Le résultat apparaît dans le fichier spécifié en paramètre ou sur la sortie standard.

Pour simplifier, on ne tient ici pas compte du format imposé par MIME sur la longueur maximale des lignes. Cela ne devrait cependant pas être le plus compliqué à gérer.

Encodage

```
#include <stdio.h>

#define b(a) (((a)>=0 && (a)<=25)?((a)+'A'):(((a)>=26 && (a)<=51)?((a)-26+'a'):(((a)>=52 && (a)<=61)?((a)-52+'0'):(((a)==62)?('+'):(((a)==63)?('/'):('='))))))

int main ( int argc, char *argv[] ) {

  FILE *f, *g;
  unsigned char buff[3];
  int i=0;

  if ( !(f = fopen(argv[1],"r"))) f = stdin ;
```

```
 if ( !(g = fopen(argv[2],"w"))) g = stdout;

 buff[0] = buff[1] = buff[2] = 0;

 while ( ( i = fread(buff,1,3,f) ) > 0 ) {
  fprintf(g,"%c",b(buff[0]>>2) );
  fprintf(g,"%c",b(((buff[0] & 0x3) << 4) | buff[1] >> 4 ) );
  if ( i > 1 ) {
    fprintf(g,"%c",b(((buff[1] & 0xF) << 2) | buff[2] >> 6 ) );
    if ( i > 2 ) {
      fprintf(g,"%c",b(buff[2] & 0x3F ) );
    } else
      fprintf(g,"=");
  } else
      fprintf(g,"==");

  buff[0] = buff[1] = buff[2] = 0;
 }

 fclose(f);
 fclose(g);

 return 0;
}
```

La compilation est classique:

```
gcc –o enc enc.c
```

et s'utilise tout aussi simplement:

```
>echo "hello world" | enc
aGVsbG8gd29ybGQK
```

Décodage

```
#include <stdio.h>

#define b(a) ((a>='A'&& a<='Z')?(a-65):\
            ((a>='a' && a<='z')?(a-71):\
             ((a>='0' && a<='9')?(a+4):\
              ((a=='+')?(62):\
               ((a=='/')?(63):(0))))))

#define isEnd(a) if ( a == '=' ) break;

int main ( int argc, char *argv[] ) {
```

```
 FILE *f, *g;
 unsigned char buff[4];

 if ( !(f = fopen(argv[1],"r"))) f = stdin ;
 if ( !(g = fopen(argv[2],"w"))) g = stdout;


 while ( fread(buff,1,4,f) == 4 ) {
  fprintf(g,"%c",b(buff[0])<<2   | b(buff[1])>>4 & 0x3 );
  isEnd(buff[2])
  fprintf(g,"%c",b(buff[1])<<4 | b(buff[2])>>2 & 0xF);
  isEnd(buff[3])
  fprintf(g,"%c",b(buff[2])<<6 | b(buff[3]) & 0x3F);
 }

 fclose(f);
 fclose(g);

 return 0;
}
```

La compilation est classique:

gcc –o dec dec.c

et s'utilise tout aussi simplement:

```
>echo aGVsbG8gd29ybGQK | dec
hello world
```

Dans la pratique

Si vous reprenez le premier exemple, vous pouvez à présent le déchiffrer facilement. Il s'agit d'un email texte avec une pièce jointe (qui se trouve être un image).

A présent, voici un petit exemple pratique un peu plus complexe :

```
From: "Nicolas" <nicolas@salemioche.com>
To: <nicolas@salemioche.servebeer.com>
Subject: un joli email en html
Date: Wed, 22 Jan 2003 19:40:51 +0100
Message-ID: <000301c2c245$ca059290$c800100a@njhome.com>
MIME-Version: 1.0
Content-Type: multipart/related;  boundary="----=_NextPart_000"

This is a multi-part message in MIME format.
```

------=_NextPart_000
Content-Type: multipart/alternative;
boundary="----=_NextPart_001"

------=_NextPart_001
Content-Type: text/plain; charset="us-ascii"
Content-Transfer-Encoding: 7bit

d'abord un peu de texte en rouge
et un image incluse : un point blanc

------=_NextPart_001
Content-Type: text/html; charset="us-ascii"
Content-Transfer-Encoding: quoted-printable

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML><HEAD><TITLE>Message</TITLE></HEAD><BODY>
<DIV><FONT
color=3D#ff0000=20
size=3D2>d'abord un peu de texte en rouge</FONT></DIV>
<DIV><FONT color=3D#ff0000 size=3D2>et un image incluse : =
<IMG=20
alt=3D"un point blanc" hspace=3D0 src=3D"cid:049384018@22012003-17E5" =
align=3Dbaseline=20
border=3D0></FONT></DIV></BODY></HTML>

------=_NextPart_001--

*------=_NextPart_000_*
Content-Type: application/octet-stream; name="Image1.jpg"
Content-Transfer-Encoding: base64
Content-ID: <049384018@22012003-17E5>

/9/ (texte encodé tronqué)…D/2Q==

------=_NextPart_000_--
.

On retrouve le MIME-Version obligatoire. Ensuite j'ai un peu triché car le Content-Type est Multipart/related, défini seulement dans la RFC 2112 d'extensions à MIME. Mais sa syntaxe reste similaire à tous les *Multipart*. On a donc 2 objets séparés par le boundary *----=_NextPart_000*. Dans le premier body part, on trouve un multipart/alternative avec encore deux objets séparés par le boundary *----=_NextPart_001*. On a ici une bonne démonstration du sous-type alternative, puisque le message est disponible en html et en mode texte. On voit aussi les possibilités récursives de MIME avec cette imbrication d'objet (body part) de type Multipart.

Le Multipart/relative permet de faire référence à un objet dans un autre. Ici, le document html (premier body part), fait référence à l'image (deuxième body part).

Pour le Coder en C

Rien de très compliquer pour inclure le support de MIME à vos applications C, en partant des codes des articles précédents sur SMTP et POP3. Pour SMTP, selon le nombre de types que vous souhaitez supporter, vous aurez plus ou moins de travail. Mais sachez qu'avec les types *text, Application/Octet-Stream et Multipart/Mixed*, vous pourrez gérer les emails courants sans difficulté.

Makemime

Pour l'écriture de scripts d'envoi d'emails complexes, il existe un outil bien pratique. Le package maildrop propose l'outil makemime (/usr/bin/makemime après une installation standard) particulièrement utile pour vous simplifier la construction de message avec MIME.

Exemple de commande:

```
-j
(
   -m
       multipart/mixed
   -a
       Mime-Version: 1.0
   (
       -j
       (
           -m
               multipart/alternative
           -a
               Content-Disposition: inline
           (
               -c
                   text/plain; charset=iso-8859-1
               msg.txt
           )
       )
       (
           -c
               text/html; charset=iso-8859-1
           msg.html
       )
   )
)
-o
   output.msg
(
   -c
       image/gif
   -a
       Content-Disposition: attachment
   attachment.gif
)
```

Vous fournissez donc les fichiers contenant les données à transférer, indiquez l'encodage et la forme du message et makemime se charge de former un message valide et de l'encodage le cas échéant. Le résultat sera proche de l'exemple brut démontré ci-dessus. Il ne vous reste donc plus qu'à ajouter une entête SMTP valide pour obtenir un email complet.

Les RFCs

MIME est un standard très complet et est donc défini par de nombreuses RFCs.

RFCs principales:

RFC 2045: MIME Part One: Format of Internet Message Bodies
RFC 2046: MIME Part Two: Media Types
RFC 2047: MIME Part Three: Message Header Extensions for Non-ASCII Text
RFC 2048: MIME Part Four: Registration Procedures
RFC 2049: MIME Part Five: Conformance Criteria and Examples

RFCs secondaires:
RFC 1524: *The formal description of mailcap files. Mailcap files describe how to handle media types.*
RFC 2015: *MIME Security with Pretty Good Privacy (PGP).*
RFC 2110: *MIME E-mail Encapsulation of Aggregate Documents, such as HTML (MHTML).*
RFC 2111: *Content-ID and Message-ID Uniform Resource Locators.*
RFC 2112: *The MIME Multipart/Related Content-type.*
RFC 2183*: Defines the syntax and sematics of the "Content-Disposition" header to convey presentational information.*
RFC 2184*: MIME Parameter Value and Encoded Word Extensions: Character Sets, Languages, and Continuations*

Conclusion

La plupart des mails que l'on reçoit sont des messages MIME, ce standard est donc devenu complètement indissociable de SMTP ou POP3. Vous pouvez à présent programmer vos applications pour qu'elles envoient des messages multimédia (mot très à la mode au moment de la parution de cette norme).

Ainsi s'achève, cette série de trois articles sur les technologies et protocoles liées au courrier électronique. J'espère qu'ils vous ont aidé à comprendre les mécanismes de cet outil et vous ont donné l'envie de jouer avec.

Nicolas JEAN

http://www.nicolasjean.com