MIME

Multipurpose Internet Mail Extensions (MIME) is an Internet standard that was originally designed to extends the format of email to support:

- Text and header information in character sets other than ASCII,
- Non-text attachments
- Message bodies with multiple parts

However, MIME's use has grown beyond describing the content of email to describe content type in general in Web applications. For more information, please refer to the MIME page definition on Wikipedia.

A MIME message is defined through several headers, such as "content-type", that indicates the media type of the message or the message part. These information are necessary for the server to interpret the incoming data.

In Wakanda, MIME features are used in the following areas:

- **HTTP Request Handlers**,
- **WAF-Mail** CommonJS module.

Note: For information about Wakanda Server built-in MIME file type support, please refer to the Mime Types Support section.

MIMEMessage

A *MIMEMessage* object contains MIME formated data, such as a multiparts form.

In Wakanda, *MIMEMessage* objects are available:

- through the **parts** property of an **HTTPRequest**. These objects gives access to the list of uploaded parts from an HTTP client in the context of a multipart form.
- through the **getMIMEMessage( )** method of the **MIMEWriter** class.

count

Description

The **count** property returns the total number of parts contained in the multipart message.

In case of a form uploaded by a HTTP client, it is the number of parts uploaded by the client.

encoding

Description

The **encoding** property returns the encoding used for the request parts.

boundary

Description

The **boundary** property returns the boundary tag value used to delimit the parts in the multipart MIME message.

[n]

Description

The **[n]** property gives access to the nth part of the *MIMEMessage*. This part is an object of the **MIMEMessagePart** class for which you have properties and a method.

length

Description

The **length** property returns the total number of parts contained in the multipart message.

In case of a form uploaded by a HTTP client, it is the number of parts uploaded by the client.

toBlob( )

Blob **toBlob**([String *mimeType*])

Parameter	Type	Description
mimeType	String	Type of the MIME message or "application/octet-stream" if omitted
Returns	Blob	MIME message as Blob

Description

The **toBlob( )** method returns the MIME message as a *Blob* object.

In the optional *mimeType* parameter, you can pass a lower case string representing the media type of the MIME message (see RFC2046). By default if you omit this parameter, the *Blob* media type is "application/octet-stream".

Pay attention to the size of manipulated objects since the method creates in memory a copy of the MIME message. An error is thrown if there is not enough memory available to execute the operation.

toBuffer( )

void **toBuffer**( )

Description

The **toBuffer( )** method returns the MIME message as a *Buffer* object.

MIMEMessagePart

MIMEMessagePart objects are the individual parts of a multipart message (**HTTPRequest** or **Mail Instances**). These objects are available through the following syntax:

```
message.messageParts[n]
```

... where *n* is the part number of the MIME message.

name

Description

The **name** property returns the name of the input field used for the POST of the binary data.

fileName

Description

The **fileName** property returns the name of the uploaded file.

mediaType

Description

The **mediaType** property returns the value of the part's "content-type" field.

size

Description

The **size** property returns the size of the body part in bytes.

asText

Description

The **asText** property returns the body of the part as a Text value. If the body contents do not match the String type, an *Undefined* value is returned.

asPicture

Description

The **asPicture** property returns the body of the part as an *Image* value if possible. If the body contents do not match the image type, an *Undefined* value is returned.

asBlob

Description

The **asBlob** property returns the body of the part as a *BLOB* value regardless of the actual data type in the body (text, image, or any other data type).

save( )

void **save**(String *filePath* [, Boolean *overWrite*])

Parameter	Type	Description
filePath	String	Path of the destination file
overWrite	Boolean	true = overwrite the destination file if it already exists

Description

The **save( )** method saves the body of the part in the file whose path is passed in *filePath*.

If *filePath* describes a full path including a filename, the given name is used for the file. Otherwise, if *filePath* only describes a folder path, the original filename (returned by the **name** property) is used.

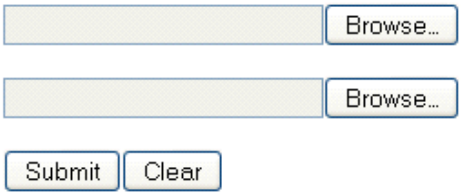
If the *overWrite* parameter value is set to true, the destination file is replaced if it already exists. If it is set to false or is omitted, the **save( )** action is ignored if the destination file already exists.

Example

The following requestHandler function displays the parts posted by a simplified HTML form:

```
function displayFormContent (request, response)
{
    var i;
    var result;

    result = 'request.parts.count: ' + request.parts.count;
    result = result + '\nrequest.parts.encoding: ' + request.parts.encoding;
    result = result + '\nrequest.parts.boundary: ' + request.parts.boundary;
    result = result + '\n-------------------------------------';

    for (i = 0; i < request.parts.count; ++i) {
        result = result + '\nrequest.parts[' + i +'].name: ' + request.parts[i].na
        result = result + '\nrequest.parts[' + i + '].fileName: ' + request.parts[
        result = result + '\nrequest.parts[' + i + '].mediaType: ' + request.parts
        result = result + '\nrequest.parts[' + i + '].size: ' + request.parts[i].s
        result = result + '\nrequest.parts[' + i + '].asText: ' + request.parts[i]
        result = result + '\n-------------------------------------';

        request.parts[i].save('e:/Data/', true);
    }

    return result;
}
```

The HTML form appears as shown below:



Here is the HTML form code:

```
<form method="post" action="/displayFormContent" enctype="multipart/form-data">
<p><input type="file" name="fileBlob1" size="25"></p>
<p><input type="file" name="fileBlob2" size="25"></p>
<p><input type="submit" value="Submit"><input type="reset" value="Clear"></p>
</form>
```

MIMEWriter

The **MIMEWriter** class allows you to create and build new MIME objects that you can convert to regular *MIMEMessage* objects using the **getMIMEMessage( )** method.

addPart( )

void **addPart**(String | Blob | Image *part* , String *name* [, String *mimeType*])

Parameter	Type	Description
part	String, Blob, Image	Part to add to the message
name	String	Part name
mimeType	String	Media type of the part

Description

The **addPart( )** method adds a new *part* to the MIME message being written.

Pass in *part* the MIME part you want to add to the MIME message. Currently, you can only pass objects of the text, blob or image type.

Pass in the *name* parameter the name for the *part*. In SMTP client applications, this name will be proposed to save the part on disk. Pass an empty string if you do not want to give a name to the *part*.

Pass the MIME media type of the part in *mimeType*.

- If you passed a *string* in *part* and *mimeType* is omitted, "text/plain" will be used as a default value.
- If you passed an *Image* in *part* and *mimeType* is omitted, "image/*xyz*" (for example "image/png") will be used as a default value.
- Otherwise, if *mimeType* is omitted, "application/octet-stream" will be used as a default value.

By default, images will be encoded in base64.

Example

With this basic handler function, your can return a multipart message in a Blob:

```
function testMIME(request, response) {
    var mimeWriter = new MIMEWriter();  // creates the message
    var photo = loadImage("c:/temp/Tulips.jpg");
    mimeWriter.addPart ("Look at these beautiful flowers\r\n", "", 'text/plain');
    mimeWriter.addPart (photo, "", 'image/jpeg');
    var mimeMessage = mimeWriter.getMIMEMessage();
    var blob = mimeMessage.toBlob ('text/plain');

    response.headers.CONTENT_TYPE = 'text/plain';

    return blob;
}
```

The returned blob looks like:

```
--A3BA45AE53BB1C4ABFF9F531A377156D
Content-Type: text/plain; name="lenom1"
Content-Transfer-Encoding: 8bit

Look at these beautiful flowers

--A3BA45AE53BB1C4ABFF9F531A377156D
Content-Type: image/png;image/jpeg;image/gif; name="lenom2"
Content-Disposition: attachment; filename="lenom2"
Content-Transfer-Encoding: base64

/9j/4AAQSkZJRgABAgEAYABgAAD/7gAOQWR (... base64 encoding)
```

getMIMEBoundary( )

String **getMIMEBoundary**( )

Returns	String	Boundary used to delimit MIME parts

Description

The **getMIMEBoundary( )** method returns the boundary string used to delimit each MIME message part. This method is useful to set an appropriate Content-type value for the messages.

getMIMEMessage( )

MIMEMessage **getMIMEMessage**( )

Returns	MIMEMessage	Regular MIME message

Description

The **getMIMEMessage( )** method converts the *MIMEWriter* current object to a valid **MIMEMessage** object.

MIMEWriter( )

void **MIMEWriter**( )

Description

The **MIMEWriter( )** method is the constructor of the dedicated class objects of the *MIMEWriter* type. It allows you to create new empty MIME messages objects on the server, that you can convert to regular multipart *MIMEMessage* objects using the **getMIMEMessage( )** method.

Example

To build a basic message:

```
function myMIME(request, response) {
    var mimeWriter = new MIMEWriter();  // creates the message
    mimeWriter.addPart ("Part 1", "", 'text/plain');
    mimeWriter.addPart ("Part 2", "", 'text/plain');
    var mimeMessage = mimeWriter.getMIMEMessage();
    var blob = mimeMessage.toBlob ('text/plain');
    response.headers.CONTENT_TYPE = 'text/plain';
return blob;
}
```