BLOB

Wakanda implements Server-side the Blobs interface. A BLOB (Binary Large OBject) contains raw data. A *Blob* instance is immutable and serves as a transport mecanism to manipulate arbitrary amount of bytes. If you need to write or modify individual bytes within a *Blob*, you must first convert it to a *Buffer* object using the **toBuffer( )** function provided by Wakanda. Note that Wakanda provides the corresponding **toBlob( )** method in the **Buffers** class.

Wakanda Blobs interface is basically compliant with the W3C Blob Interface, describing the **size** and **type** properties as well as the **slice( )** method.

Additional methods such as **toBuffer( )** and **toString( )** are provided for specific Wakanda needs.

In addition to the **Blob( )** constructor method, Blob objects can be created and handled in Wakanda through the following features:

- the **body** property of the *HTTPRequest* objects
- the **asBlob** property of the *HTMLFormParts* objects
- the **body** property of the *HTTPResponse* objects
- the **sendChunkedData( )** method of the *HTTPResponse* objects
- the **toBlob( )** method of the *Buffer* objects
- the **Storage Attribute Types** in the Datastore Class designer

***Note**: In compliance with the W3C File API specification for HTML5, Wakanda **File** objects inherit from the **BLOB** class (Feature available in the Develoment branch only).*

BLOB Class Constructor

Blob( )

void **Blob**(Number *size* [, Number *filler*] [, String *mimeType*])

Parameter	Type	Description
size	Number	Size of the new Blob in bytes
filler	Number	Filler character code value
mimeType	String	Media type of the Blob

Description

The **Blob( )** method is the constructor of the class objects of type *Blob*. It allows you to create new BLOB objects on the server.

Pass in *size* the expected size of the *Blob* in memory. The size must be expressed in bytes.

If you want to initialize each byte of the *Blob* to a specific character, pass the corresponding character code into the *filler* optional parameter. For example, pass 88 to fill the BLOB with "X". By default if you omit this parameter, the *Blob* is filled with "0" (zeros).

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

Example

We want to create a 20 bytes Blob, filled with X and associated to the standard binary MIME type:

```
var myBlob = new Blob( 20 , 88, "application/octet-stream");
var myString = myBlob.toString();
//myString contains "XXXXXXXXXXXXXXXXXXXX"
```

BLOB Instances

size

Description

The **size** property returns the size of the *Blob* expressed in bytes.

type

Description

The **type** property returns the media type of the *Blob*, expressed as a MIME type. If the MIME type of the *Blob* is unknown, the property value is an empty string.

copyTo( )

void **copyTo**(File | String *destination* [, Boolean | String *overwrite*])

Parameter	Type	Description
destination	File, String	Destination file
overwrite	Boolean, String	True or "Overwrite" to override existing file if any, otherwise False or "KeepExisting"

Description

The **copyTo( )** method copies the Blob referenced in the *BLOB* object (the source object) into the specified *destination* file.

Usually, the BLOB object will contain a *File*. The file referenced in the source object must already exist, otherwise the method returns a "File not found" error.

In the *destination* parameter, you can pass a *File* object or a string containing an absolute path or a URL to a file.

By default, a "File already exists" error will occur if there is a file with the same name as the source file at the defined *destination*. You can change this behavior by using the *overwrite* parameter:

- If you pass *True* or the "OverWrite" string in *overwrite*, the method will delete and override the existing file without any error.
- If you pass *False* or the "KeepExisting" string in *overwrite* (or omit the parameter), the existing file is left untouched and an error is generated. By default, the file is not overwritten.

Example

The following example duplicates a file in its own folder:

```
var myFile = new File ("c:/Documents/Invoice.txt") ; // get the File object and
myFile.copyTo ( "c:/Documents/Invoice_copy.txt") ; // duplicate it to the same location
```

Example

The following example duplicates a file and overwrites the copy if it already exists:

```
var aCopy = new File ("c:/Documents/Invoice_copy.txt") ;
if (aCopy.exists)
    {
        var myFile = new File ("c:/Documents/Invoice2011.txt") ;
        myFile.copyTo ( "c:/Documents/Invoice_copy.txt" , "OverWrite") ;
    }
```

slice( )

Blob **slice**([Number *start* [, Number *end*[, String *mimeType*]]])

Parameter	Type	Description
start	Number	Starting byte in the Blob
end	Number	Last byte to get
mimeType	String	Media type of the Blob
Returns	Blob	New Blob referencing bytes

Description

The **slice( )** method creates a new *Blob* object by referencing the contents of the bytes of the *Blob* to which it is applied, from *start* to *end*.

In *start*, you can pass the position of the first byte to get in the *Blob* (starting a 0) and in end, you can pass the position of the last byte to get in the *Blob*. In other words, the length of the returned Blob is *end* - *start*.

- If *start* is omitted, the **slice( )** operation starts at position 0 by default.
- If *end* is omitted, the **size** value is used by default.
- You can pass a negative value in *end* or *start* to designate a position evaluated from the last byte of the *Blob*.

In the optional *mimeType* parameter, you can pass a lower case string representing the media type of the new Blob, expressed as a MIME type (see RFC2046). By default if you omit this parameter, the *Blob* media type is "" (empty string).

Keep in mind that, since the new Blob is a reference to a part of the source Blob, any modification of its contents will also affect the source, and conversely.

Blob slices are very useful when performance is an issue, since they avoid copying and duplicating data by simply designating a byte range. However, they should be handled with care because modifications in one Blob can affect other ones.

Example

```
var myBlob = new Blob( 20 , 88, "application/octet-stream");
var myString = myBlob.toString();
// myString contains "XXXXXXXXXXXXXXXXXXXX"
var myb = myBlob.slice(5,10);
myString = myb.toString();
// myString contains "XXXXX"
```

Example

As *File* objects inherit from Blobs, you can read bytes from a file:

```
var foo = File("c:/temp/test.txt");
foo.slice(0,100).toString(); // read the 100 first bytes as text
```

toBuffer( )

Buffer **toBuffer**( )

Returns	Buffer	Buffer object containing a copy of the Blob

Description

The **toBuffer( )** method returns a *Buffer* object containing a copy of the *Blob* bytes. Once the original bytes are converted to a *Buffer* object, you can write or edit them using the various methods and properties of the **Buffer Instances** class. This step is necessary when you want to process Blobs stored as attributes of a Datastore class (see. **Storage Attribute Types**).

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

***Note:** The **toBuffer( )** method is not part of the W3C Blob Interface specification.*

Example

We convert a Blob to Buffer to be able to edit its contents, and then convert it back to Blob:

```
var myBlob = new Blob( 20 , 88, "application/octet-stream");
var myString = myBlob.toString(); // myString contains "XXXXXXXXXXXXXXXXXXXX"
var tempBuf = myBlob.toBuffer(); // we have a buffer object now
    // we can call buffer edition methods such as write()
var nbbytes = tempBuf.write("Hello World!",4);
myBlob = tempBuf.toBlob(); // convert back to blob
myString = myBlob.toString();
    // myString contains "XXXXHello World!XXXX"
```

toString( )

String **toString**([String *stringFormat*])

Parameter	Type	Description

stringFormat	String	IANA encoding of the string
Returns	String	Blob expressed as string

Description

The **toString( )** method allows you to get a string representation of the *Blob* contents.

Pass in the optional *stringFormat* parameter a string indicating the character encoding to use for the *Blob* contents interpretation. This parameter should be expressed as a standard IANA name of the set, for example "ISO-8859-1" or "utf-8" (for more information, refer to the IANA character sets document). By default, if the *stringFormat* parameter is omitted, "utf-8" is used.

***Note**: The **toString( )** method is not part of the W3C Blob Interface specification.*

Example

We want to create a 20 bytes Blob, filled with X and associated to the standard binary MIME type:

```
var myBlob = new Blob( 20 , 88, "application/octet-stream");
var myString = myBlob.toString();
//myString contains "XXXXXXXXXXXXXXXXXXXX"
```