

# PatchRequest



Apr 11, 2024 10:10

Supported Clients

SmartClient WebClient NGClient

Methods Summary		
Boolean	<code>addFile(parameterName, jsFile)</code>	Add a file to the post; it will try to get the correct mime type from the file name or the first bytes.
Boolean	<code>addFile(parameterName, jsFile, mimeType)</code>	Add a file to the post with a given mime type; could also be used to force the default 'application/octet-stream' on it, because this plugin will try to guess the correct mime type for the given file otherwise (based on the name or the bytes).
Boolean	<code>addFile(parameterName, fileName, jsFile)</code>	Add a file to the post; it will try to get the correct mime type from the file name or the first bytes.
Boolean	<code>addFile(parameterName, fileName, jsFile, mimeType)</code>	Add a file to the post with a given mime type; could also be used to force the default 'application/octet-stream' on it, because this plugin will try to guess the correct mime type for the given file otherwise (based on the name or the bytes).
Boolean	<code>addFile(parameterName, fileName, fileLocation)</code>	Add a file to the post; it will try to get the correct mime type from the file name or the first bytes.
Boolean	<code>addFile(parameterName, fileName, fileLocation, mimeType)</code>	Add a file to the post with a given mime type; could also be used to force the default 'application/octet-stream' on it, because this plugin will try to guess the correct mime type for the given file otherwise (based on the name or the bytes).
Boolean	<code>addHeader(headerName, value)</code>	Add a header to the request.
Boolean	<code>addParameter(name, value)</code>	Add a parameter to the post.
void	<code>executeAsyncRequest(username, password, workstation, domain, successCallbackMethod, errorCallbackMethod)</code>	Execute the request method asynchronous.
void	<code>executeAsyncRequest(username, password, workstation, domain, successCallbackMethod, errorCallbackMethod, callbackExtraArgs)</code>	Execute the request method asynchronous using windows authentication.
void	<code>executeAsyncRequest(username, password, successCallbackMethod, errorCallbackMethod)</code>	Execute the request method asynchronous.
void	<code>executeAsyncRequest(username, password, successCallbackMethod, errorCallbackMethod, callbackExtraArgs)</code>	Execute the request method asynchronous using windows authentication.
void	<code>executeAsyncRequest(successCallbackMethod, errorCallbackMethod)</code>	Execute the request method asynchronous.
void	<code>executeAsyncRequest(successCallbackMethod, errorCallbackMethod, callbackExtraArgs)</code>	Execute the request method asynchronous using windows authentication.
Response	<code>executeRequest()</code>	Execute the request method.
Response	<code>executeRequest(userName, password)</code>	Execute the request method.
Response	<code>executeRequest(userName, password, workstation, domain)</code>	Execute a request method using windows authentication.
void	<code>forceMultipart(forceMultipart)</code>	Force this request to prepare a "Content-Type: multipart/form-data" formatted message even if only one file or only a number of parameter were added to it.
void	<code>setBodyContent(content)</code>	Set the body of the request.
void	<code>setBodyContent(content, mimeType)</code>	Set the body of the request and content mime type.
void	<code>setCharset(charset)</code>	Set the charset used when posting.
void	<code>usePreemptiveAuthentication(b)</code>	Whatever to use preemptive authentication (sending the credentials in the header, avoiding the server request to the client - useful when uploading files, as some http servers would cancel the first request from the client, if too big, as the authentication request to the client was not yet sent)

Methods Details

**addFile(parameterName, jsFile)**

Add a file to the post; it will try to get the correct mime type from the file name or the first bytes.

If you add a single file then this will be a single file (so not a multi-part) post. If you want/need multi-part then you have to either add multiple files or both a file and one or more parameters using `addParameter(...)`.

Parameters

String parameterName;  
Object jsFile;

**Returns**[Boolean](#)**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
poster.addFile('myFileParamName','manual.doc','c:/temp/manual_01a.doc')
poster.addFile(null,'postXml.xml','c:/temp/postXml.xml') // sets the xml to post

var f = plugins.file.convertToJSFile('./somefile02.txt')
if (f && f.exists()) poster.addFile('myTxtFileParamName','somefile.txt', f)

f = plugins.file.convertToJSFile('./anotherfile_v2b.txt')
if (f && f.exists()) poster.addFile('myOtherTxtFileParamName', f)
```

**addFile(parameterName, jsFile, mimeType)**

Add a file to the post with a given mime type; could also be used to force the default 'application/octet-stream' on it, because this plugin will try to guess the correct mime type for the given file otherwise (based on the name or the bytes).

If you add a single file then this will be a single file (so not a multi-part) post. If you want/need multi-part then you have to either add multiple files or both a file and one or more parameters using addParameter(...).

**Parameters**

[String](#) parameterName ;  
     me  
[Object](#) jsFile ;  
[String](#) mimeType     The mime type that must be used could be the real default ('application/octet-stream') if the files one (by name or by its first bytes) is not good.

**Returns**[Boolean](#)**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
poster.addFile('myFileParamName','manual.doc','c:/temp/manual_01a.doc', 'application/msword')
poster.addFile(null,'postXml.xml','c:/temp/postXml.xml', 'text/xml') // sets the xml to post

var f = plugins.file.convertToJSFile('./somefile02.txt')
if (f && f.exists()) poster.addFile('myTxtFileParamName','somefile.txt', f, 'text/plain')

f = plugins.file.convertToJSFile('./anotherfile_v2b.txt')
if (f && f.exists()) poster.addFile('myOtherTxtFileParamName', f, 'text/plain')
```

**addFile(parameterName, fileName, jsFile)**

Add a file to the post; it will try to get the correct mime type from the file name or the first bytes.

If you add a single file then this will be a single file (so not a multi-part) post. If you want/need multi-part then you have to either add multiple files or both a file and one or more parameters using addParameter(...).

**Parameters**

[String](#) parameterName;  
[String](#) fileName ;  
[Object](#) jsFile ;

**Returns**[Boolean](#)**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
poster.addFile('myFileParamName','manual.doc','c:/temp/manual_01a.doc')
poster.addFile(null,'postXml.xml','c:/temp/postXml.xml') // sets the xml to post

var f = plugins.file.convertToJSFile('./somefile02.txt')
if (f && f.exists()) poster.addFile('myTxtFileParamName','somefile.txt', f)

f = plugins.file.convertToJSFile('./anotherfile_v2b.txt')
if (f && f.exists()) poster.addFile('myOtherTxtFileParamName', f)
```

**addFile(parameterName, fileName, jsFile, mimeType)**

Add a file to the post with a given mime type; could also be used to force the default 'application/octet-stream' on it, because this plugin will try to guess the correct mime type for the given file otherwise (based on the name or the bytes).

If you add a single file then this will be a single file (so not a multi-part) post. If you want/need multi-part then you have to either add multiple files or both a file and one or more parameters using addParameter(...).

**Parameters**

**String** parameterName ;  
     me  
**String** fileName ;  
**Object** jsFile ;  
**String** mimeType The mime type that must be used could be the real default ('application/octet-stream') if the file's one (by name or by its first bytes) is not good.

**Returns**

**Boolean**

**Supported Clients**

SmartClient, WebClient, NGClient

**Sample**

```
poster.addFile('myFileParamName','manual.doc','c:/temp/manual_01a.doc', 'application/msword')
poster.addFile(null,'postXml.xml','c:/temp/postXml.xml', 'text/xml') // sets the xml to post

var f = plugins.file.convertToJSFile('./somefile02.txt')
if (f && f.exists()) poster.addFile('myTxtFileParamName','somefile.txt', f, 'text/plain')

f = plugins.file.convertToJSFile('./anotherfile_v2b.txt')
if (f && f.exists()) poster.addFile('myOtherTxtFileParamName', f, 'text/plain')
```

**addFile(parameterName, fileName, fileLocation)**

Add a file to the post; it will try to get the correct mime type from the file name or the first bytes.

If you add a single file then this will be a single file (so not a multi-part) post. If you want/need multi-part then you have to either add multiple files or a file and at least a parameter via addParameter(...).

**Parameters**

**String** parameterName ;  
**String** fileName ;  
**String** fileLocation ;

**Returns**

**Boolean**

**Supported Clients**

SmartClient, WebClient, NGClient

**Sample**

```
poster.addFile('myFileParamName','manual.doc','c:/temp/manual_01a.doc')
poster.addFile(null,'postXml.xml','c:/temp/postXml.xml') // sets the xml to post

var f = plugins.file.convertToJSFile('./somefile02.txt')
if (f && f.exists()) poster.addFile('myTxtFileParamName','somefile.txt', f)

f = plugins.file.convertToJSFile('./anotherfile_v2b.txt')
if (f && f.exists()) poster.addFile('myOtherTxtFileParamName', f)
```

**addFile(parameterName, fileName, fileLocation, mimeType)**

Add a file to the post with a given mime type; could also be used to force the default 'application/octet-stream' on it, because this plugin will try to guess the correct mime type for the given file otherwise (based on the name or the bytes).

If you add a single file then this will be a single file (so not a multi-part) post. If you want/need multi-part then you have to either add multiple files or both a file and one or more parameters using addParameter(...).

**Parameters**

`String` parameterName ;  
     me  
`String` fileName ;  
`String` fileLocation ;  
`String` mimeType The mime type that must be used could be the real default ('application/octet-stream') if the files one (by name or by its first bytes) is not good.

**Returns**

`Boolean`

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
poster.addFile('myFileParamName','manual.doc','c:/temp/manual_01a.doc', 'application/msword')
poster.addFile(null,'postXml.xml','c:/temp/postXml.xml', 'text/xml') // sets the xml to post

var f = plugins.file.convertToJSFile('./somefile02.txt')
if (f && f.exists()) poster.addFile('myTxtFileParamName','somefile.txt', f, 'text/plain')

f = plugins.file.convertToJSFile('./anotherfile_v2b.txt')
if (f && f.exists()) poster.addFile('myOtherTxtFileParamName', f, 'text/plain')
```

**addHeader(headerName, value)**

Add a header to the request.

**Parameters**

`String` headerName;  
`String` value ;

**Returns**

`Boolean`

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
method.addHeader('Content-type','text/xml; charset=ISO-8859-1')
```

**addParameter(name, value)**

Add a parameter to the post.

If there is also at least one file added to this request using addFile(...) then a multi-part post will be generated.

**Parameters**

`String` name ;  
`String` value ;

**Returns**

`Boolean`

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
poster.addParameter('name','value')
poster.addParameter(null,'value') //sets the content to post
```

**executeAsyncRequest(username, password, workstation, domain, successCallbackMethod, errorCallbackMethod)**

Execute the request method asynchronous. Success callback method will be called when response is received. Response is sent as parameter in callback.  
 This Response can be a response with a different status code then just 200, it could also be 500, which is still a valid response from the server, this won't go into the error callback.  
 So you need to test the `Reponse.getStatusCode()` for that to know if everything did go OK.  
 If no response is received (request errors out), the errorCallbackMethod is called with exception message as parameter.

**Parameters**

<code>String</code>	username	the user name
<code>String</code>	password	the password
<code>String</code>	workstation	The workstation the authentication request is originating from.
<code>String</code>	domain	The domain to authenticate within.
<code>Function</code>	successCallbackMethod	callbackMethod to be called after response is received
<code>Function</code>	errorCallbackMethod	callbackMethod to be called if request errors out

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
method.executeAsyncRequest('username','password','mycomputername','domain',globals.successCallback,globals.errorCallback)
```

**executeAsyncRequest(username, password, workstation, domain, successCallbackMethod, errorCallbackMethod, callbackExtraArgs)**

Execute the request method asynchronous using windows authentication.  
 Success callback method will be called when response is received. Response is sent as parameter in callback followed by any 'callbackExtraArgs' that were given.  
 This Response can be a response with a different status code then just 200, it could also be 500, which is still a valid response from the server, this won't go into the error callback.  
 So you need to test the `Reponse.getStatusCode()` for that to know if everything did go OK.  
 If no response is received (request errors out, network errors), the errorCallbackMethod is called with exception message as parameter followed by any 'callbackExtraArgs' that were given.

**Parameters**

<code>String</code>	username	the user name
<code>String</code>	password	the password
<code>String</code>	workstation	The workstation the authentication request is originating from.
<code>String</code>	domain	The domain to authenticate within.
<code>Function</code>	successCallbackMethod	callbackMethod to be called after response is received
<code>Function</code>	errorCallbackMethod	callbackMethod to be called if request errors out
<code>Array</code>	callbackExtraArgs	extra arguments that will be passed to the callback methods; can be used to identify from which request the response arrived when using the same callback method for multiple requests. Please use only simple JSON arguments (primitive types or array/objects of primitive types)

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
method.executeAsyncRequest('username','password','mycomputername','domain',globals.successCallback,globals.errorCallback, [callIDInt])
```

**executeAsyncRequest(username, password, successCallbackMethod, errorCallbackMethod)**

Execute the request method asynchronous. Success callback method will be called when response is received. Response is sent as parameter in callback.  
 This Response can be a response with a different status code then just 200, it could also be 500, which is still a valid response from the server, this won't go into the error callback.  
 So you need to test the Reponse.getStatusCode() for that to know if everything did go OK.  
 If no response is received (request errors out), the errorCallbackMethod is called with exception message as parameter.

**Parameters**

String	username	the user name
String	password	the password
Function	successCallbackMethod	callbackMethod to be called after response is received
Function	errorCallbackMethod	callbackMethod to be called if request errors out

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
method.executeAsyncRequest(globals.successCallback,globals.errorCallback)
```

**executeAsyncRequest(username, password, successCallbackMethod, errorCallbackMethod, callbackExtraArgs)**

Execute the request method asynchronous using windows authentication.  
 Success callback method will be called when response is received. Response is sent as parameter in callback followed by any 'callbackExtraArgs' that were given.  
 This Response can be a response with a different status code then just 200, it could also be 500, which is still a valid response from the server, this won't go into the error callback.  
 So you need to test the Reponse.getStatusCode() for that to know if everything did go OK.  
 If no response is received (request errors out, network errors), the errorCallbackMethod is called with exception message as parameter followed by any 'callbackExtraArgs' that were given.

**Parameters**

String	username	the user name
String	password	the password
Function	successCallbackMethod	callbackMethod to be called after response is received
Function	errorCallbackMethod	callbackMethod to be called if request errors out
Array	callbackExtraArgs	extra arguments that will be passed to the callback methods; can be used to identify from which request the response arrived when using the same callback method for multiple requests. Please use only simple JSON arguments (primitive types or array/objects of primitive types)

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
method.executeAsyncRequest(globals.successCallback,globals.errorCallback, [callIDInt])
```

**executeAsyncRequest(successCallbackMethod, errorCallbackMethod)**

Execute the request method asynchronous. Success callback method will be called when response is received. Response is sent as parameter in callback.  
 This Response can be a response with a different status code then just 200, it could also be 500, which is still a valid response from the server, this won't go into the error callback.  
 So you need to test the Reponse.getStatusCode() for that to know if everything did go OK.  
 If no response is received (request errors out), the errorCallbackMethod is called with exception message as parameter.

**Parameters**

Function	successCallbackMethod	callbackMethod to be called after response is received
Function	errorCallbackMethod	callbackMethod to be called if request errors out

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
method.executeAsyncRequest(globals.successCallback,globals.errorCallback)
```

**executeAsyncRequest(successCallbackMethod, errorCallbackMethod, callbackExtraArgs)**

Execute the request method asynchronous using windows authentication.

Success callback method will be called when response is received. Response is sent as parameter in callback followed by any 'callbackExtraArgs' that were given.

This Response can be a response with a different status code then just 200, it could also be 500, which is still a valid response from the server, this won't go into the error callback.

So you need to test the Reponse.getStatusCode() for that to know if everything did go OK.

If no response is received (request errors out, network errors), the errorCallbackMethod is called with exception message as parameter followed by any 'callbackExtraArgs' that were given.

**Parameters**

[Fun](#) successCal callbackMethod to be called after response is received  
[ction](#)backMethod

[Fun](#) errorCallback callbackMethod to be called if request errors out  
[ction](#)ckMethod

[Arr](#) callbackExt extra arguments that will be passed to the callback methods; can be used to identify from which request the response arrived when  
[ay](#) raArgs using the same callback method for multiple requests. Please use only simple JSON arguments (primitive types or array/objects of primitive types)

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
method.executeAsyncRequest(globals.successCallback,globals.errorCallback, [callIDInt])
```

**executeRequest()**

Execute the request method.

**Returns**

[Response](#)

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
var response = method.executeRequest()
```

To be able to reuse the client, the response must be closed if the content is not read via getResponseBody or getMediaData:

```
response.close()
```

**executeRequest(userName, password)**

Execute the request method.

**Parameters**

[String](#) userName the user name

[String](#) password the password

**Returns**

[Response](#)

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
var response = method.executeRequest()

To be able to reuse the client, the response must be
closed if the content is not read via getResponseBody
or getMediaData:

response.close()
```

**executeRequest(userName, password, workstation, domain)**

Execute a request method using windows authentication.

**Parameters**

`String` `userName` the user name  
`String` `password` the password  
`String` `workstation` The workstation the authentication request is originating from.  
`String` `domain` The domain to authenticate within.

**Returns**

`Response`

**Supported Clients**

SmartClient, WebClient, NGClient

**Sample**

```
var response = method.executeRequest('username','password','mycomputername','domain');
```

**forceMultipart(forceMultipart)**

Force this request to prepare a "Content-Type: multipart/form-data" formatted message even if only one file or only a number of parameter were added to it.

It is useful because some servers require this (they only support multipart - even if you don't need to send multiple things).  
 Before Servoy 2021.03 you could force it to send multipart by adding a dummy parameter together with a single file (or the other way around) - if the server didn't object to that dummy content...

Default value: false. (if you only add one file or only parameters it will not generate a multipart request)

**Parameters**

`Boolean` `forceMultipart` if true, this request will send a multipart/form-data message even if you only add one file or only parameters. If false (default) it will send multipart only in case of multiple files or one file plus at least one parameter.

**Supported Clients**

SmartClient, WebClient, NGClient

**Sample****setBodyContent(content)**

Set the body of the request.

**Parameters**

`String` `content`;

**Supported Clients**

SmartClient, WebClient, NGClient

**Sample**

```
method.setBodyContent(content)
```

**setBodyContent(content, mimeType)**

Set the body of the request and content mime type.

**Parameters**

`String` `content` ;  
`String` `mimeType` ;



---

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
method.setBodyContent(content, 'text/xml')
```

**setCharset(charset)**

Set the charset used when posting. If this is null or not called it will use the default charset (UTF-8).

**Parameters**

[Object](#) charset;

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**

```
var client = plugins.http.createNewHttpClient();
var poster = client.createPostRequest('https://twitter.com/statuses/update.json');
poster.addParameter('status',scopes.globals.textToPost);
poster.addParameter('source','Test Source');
poster.setCharset('UTF-8');
var httpCode = poster.executeRequest(scopes.globals.twitterUserName, scopes.globals.twitterPassword).
getStatusCode() // httpCode 200 is ok
```

**usePreemptiveAuthentication(b)**

Whatever to use preemptive authentication (sending the credentials in the header, avoiding the server request to the client - useful when uploading files, as some http servers would cancel the first request from the client, if too big, as the authentication request to the client was not yet sent)

**Parameters**

[Boolean](#) b;

**Supported Clients**

SmartClient,WebClient,NGClient

**Sample**