M-Files Web Service

Documentation

Version: 1
Author: Mikko Rantanen

Contents

1. M-Files Web Service

M-Files Web Service allows programmatic access to M-Files through a REST-like interface. The service provides basic read/write access to M-Files which includes reading and modifying objects and reading document vault structures.

Unlike the traditional M-Files API, which requires ActiveX/COM support, the web service can be used from any application that can perform HTTP requests. These include .NET applications limited by code access security, applications running on non-Windows platforms and web applications written in JavaScript.

1.1 This document

This document is aimed at developers interested in using M-Files Web Service for application development. The document is divided in the following chapters:

Overview describes the general structure of M-Files Web Service.

Getting Started walks you through the basic steps required in using M-Files Web Service. The section contains examples on authentication, reading contents from M-Files as well as editing them.

Request parameters contains a description of common parameters handled by the M-Files Web Service request handler. These parameters are usable with most of the resources.

Resource reference contains description of all the resources provided by the M-Files Web Service.

Type reference contains description of all the types used to represent the resources.

Enumeration reference Lists the different enumeration types used by the M-Files Web Service.

Encoding syntax contains a reference on the more complex encoding formats used by the M-Files Web Service.

Error reporting describes the way M-Files Web Service communicates server-side exception back to the application over the HTTP protocol.

Compatibility lists best practices that enable applications to consume M-Files Web Services hosted on top of different server configurations.

Sample code contains a package of code samples to help in application development.

2. Overview

M-Files Web Service consists of several resources and the types that represent these. Most of the resource belong to one of the five major resource hierarchies. These hierarchies are listed below.

The **objects hierarchy** contains the resources used to read and edit individual objects. This includes searching objects, reading and editing metadata, downloading files and creating new objects.

The **views hierarchy** contains the resources representing the document vault view hierarchy. These resources enable applications to navigate through M-Files views. These resources are read-only as M-Files Web Service doesn't currently support creating or modifying views.

The **vault structure hierarchy** provides information on the metadata structure of the vault. The hierarchy contains resources for object types, property definitions and workflows for example. Similar to the views hierarchy, the structure hierarchy doesn't currently support modification either.

The **server hierarchy** contains the resources representing the server-state. The resources in this hierarchy are also accessible without a document vault level authentication. Some, such as /server/authenticationtokens, can even be accessed without any authentication at all.

The **session hierarchy** contains information on the current session. This hierarchy is required for Cookie-based authentication as it provides the /session and /session/vault resources.

3. Getting Started

This section demonstrates the use of M-Files Web Service by using low level HTTP requests. The information below should be applicale to most programming languages. See Sample Code section for sample M-Files Web Service C♯ client. The Sample Code package also contains the struct definitions used in the samples below.

3.1 Prerequisities

There are three prerequisities for using MFWS: A HTTP client, MFWS URL and M-Files credentials. While the last item isn't strictly required, most of the M-Files operations require authentication and access to a document vault.

The HTTP client depends on the programming platform and doesn't necessarily mean a web browser. WebClient is available for .NET framework and HttpURLConnection can be used in Java. Browser applications can perform the HTTP requests using the raw XMLHttpRequest or a wrapper around this such as jQuery's $.ajax. If such HTTP API isn't available a raw network socket API will do as well.

The MFWS URL and the credentials are M-Files installation specific. The easiest way to acquire the URL is to configure M-Files Web Access, which comes with the web service. If M-Files Web Access is configured at http://example.org/m-files, the MFWS URL is http://example.org/m-files/REST. For the credentials please contact your M-Files administrator.

3.2 Authenticating

MFWS provides three ways to authenticate the requests: Credentials in HTTP headers, cookie-based session and authentication tokens. Passing the credentials in the HTTP headers is the easiest way but sacrifices some security as the plain text credentials are passed in each request. Authentication tokens improve upon this by encrypting the credentials using asymmetric encryption. Cookie-based sessions offer some benefits in browser environments where the browser can manage the cookies. The choice of authentication mechanism has no effect on the actual use of the web service. The examples will mostly use the token based authentication.

3.2.1 Example: Acquiring the authentication token

An authentication token request is one of the requests that can be made unauthenticated. It is made by POSTing the authentication credentials to the /server/authenticationtokens resource as shown in the example 3.1 below.

```
1
2  var getToken = function (username, password, vault) {
3
4      // Request an encrypted token with the login information.
5      $.ajax({
6          url: "http://example.org/REST/server/authenticationtokens.aspx",
7          type: "POST",
8          dataType: "json",
9          contentType: "application/json",
10         data: JSON.stringify({ Username: username, Password: password,
               VaultGuid: vault }),
11         success: processToken
12     });
13 };
14
15 var processToken = function (token) {
16     // Set the header.
17     $.ajaxSetup({ headers: { "X-Authentication" : token.Value } });
18 };
```

Listing 3.1: *Requesting the authentication token.*

Once the authentication token has been acquired it can be used in the requests by passing it in the

X-Authentication header. Note authentication can be done with or without the vault GUID. Authenticating without the vault GUID allows access to the server-level resources such as the document vault collection. A vault-level authentication is required to access the information within a document vault.

3.3 Reading the resources

Once the authentication has been performed it is possible to access information within a document vault. This information is made available in terms of resources. Save for few exceptions all of these resources support the HTTP GET method which allows an application to request the current representation of the resource. The example 3.2 below uses a GET request on the document vault root view resource found at /views/items to list all the views in the document vault root.

```
1
2  var getRootItems = function () {
3
4      // Post the object data.
5      $.ajax({
6          url: "http://example.org/REST/views/items.aspx",
7          type: "GET",
8          dataType: "json",
9          success: processView
10     });
11 };
12
13 var processView = function (folderContents) {
14     $.each( folderContents, function (i, item) {
15
16         // Ignore this if the item is not a view.
17         if( item.FolderContentItemType !== 1 ) return;
18
19         alert( item.View.Name );
20     });
21 };
```

Listing 3.2: *Requesting the contents of the document vault root view.*

Testing resource reading is easy with the browser itself. You can log in to M-Files Web Access to establish the session. After this the GET-resources can be read with normal HTTP requests. However some browsers demand XML representation which MFWS is currently unable to provide for all resources. At least Internet Explorer 9 is able to read the resources in JSON format.

3.4 Writing the resources

Just like reading resources, modifying them by deleting or editing existing ones or creating new ones is done with basic HTTP requests as well. Each of these different operations has its own HTTP verb: POST for creating, PUT for editing and DELETE for deleting. However unlike GET, these other verbs are not available for all resources. The resource reference contains information on the resources and the methods available for them. The example 3.3 below uses POST request on the documents collection resource found at /objects/0 to create a new document in the document vault.

Especially older IIS versions (5.1, 6.0) make it harder to use non GET or POST verbs with ASP.Net applications. While M-Files Web Service supports pure PUT and DELETE requests, the default IIS configuration for M-Files Web Access doesn't enable PUT and DELETE requests for IIS. For this reason M-Files Web Service supports a `?_method=VERB` query parameter which is the recommended way to communicate the PUT and DELETE intents. See compatibility for more information on this. Example 3.4 below shows a PUT request on the check-out state resource.

```
var createObject = function () {

    // Post the object data.
    $.ajax({
        url: "http://example.org/REST/objects/0.aspx",
        type: "POST",
        dataType: "json",
        contentType: "application/json",
        data: JSON.stringify({
            PropertyValues: [{
                    // Document name
                    PropertyDef: 0,
                    TypedValue: { DataType: 1, Value: "Invoice" }
                }, {
                    // "Single File" property
                    PropertyDef: 22,
                    TypedValue: { DataType: 8, Value: false }
                }, {
                    // Class.
                    PropertyDef: 100,
                    TypedValue: { DataType: 9, Lookup: { Item: 0 } }
                }],
            Files: []
        }),
        success: processDocument
    });
};

var processDocument = function (objectVersion) {
    alert( objectversion.Title + " created successfully." );
};
```

Listing 3.3: *Creating a new document.*

```
var checkOut = function (type, id, version) {

    // Construct the URL.
    // .../REST/objects/(type)/(id)/(version)/checkedout?_method=PUT
    var url = "http://example.org/REST/objects/" +
              [type, id, version].join("/") +
              "/checkedout.aspx?_method=PUT";

    // Request an encrypted token with the login information.
    $.ajax({
        url: url, type: "POST", dataType: "json",
        contentType: "application/json",
        data: JSON.stringify({ Value: 2 /* Checked out to me */ }),
        success: modifyObject
    });
};

var modifyObject = function (objectVersion) {
    // Object is checked out.
};
```

Listing 3.4: *Performing a check out.*

4. Request parameters

In addition to the resource specific parameters there are few request parameters that affect the way M-Files Web Service handles the requests with all resources.

4.1 Authentication

Almost all of the M-Files Web Service resources require either an application-level or vault-level authentication. There are three alternate ways to authenticate the requests: Two header-based approaches using either plain text credentials or encrypted authentication token or a cookie based approach using the /session and /session/vault resources.

The headers used in the header-based authentication are listed below in table 4.1.

Table 4.1: *Authentication headers*

Header	*Description*
X-Username	Plain text username.
X-Password	Plain text password.
X-WindowsUser	Value of `true` if M-Files should authenticate the username and password against Windows domain. Omit this header or use the value of `false` to authenticate the user as M-Files user.
X-Domain	Windows domain for Windows authentication. The header value can be empty or the header can be omitted to use the default domain.
X-Authentication	Encrypted authentication token. This can be requested through /server/authenticationtokens and /session/authenticationtoken resources.
X-Vault	Document vault GUID. Application-level resources such as the document vault list can be accessed even if this header is missing.
X-ComputerName	Unique identifier for the client computer. This is used mainly to distinguish check-outs from different computers. This header always overrides the computer name - even if an authentication token containing a computer name is used.

The authentication token can be created with or without the vault-information. If the token is created without the vault-information it provides only an application-level token. Application-level authentication token can be used to request the vault listing from the server in which case the listing contains vault-level authentication tokens or alternatively the application-level authentication token may be combined with the `X-Vault` header to acquire a vault-level authentication context.

M-Files Web Service also supports passing the header values as query parameters in case the header values cannot be modified for one reason or another. The mapping between headers and query parameters is listed in table 4.2 below. In case a parameter is defined both as a header and as a query parameter the header value is used.

Table 4.2: *Authentication headers*

Header	*Query parameter*
X-Username	username
X-Password	password
X-WindowsUser	windowsuser
X-Domain	domain
X-Authentication	auth
X-Vault	vault
X-ComputerName	computername

4.2 Content type negotiation

HTTP protocol defines two headers that are used to negotiate the content types used within the requests and the responses. These are the `Content-Type` and `Accept` headers. `Content-Type` header describes the type of content transmitted in the request or response body while the `Accept` header is sent by the client to the server to inform the server what kind of content the client prefers.

M-Files Web Service acknowledges this header and currently the only supported value for it is `application/json` save for few resource-specific exceptions. These exceptions are mainly in the resources that accept or respond with file contents. There is experimental support for `application/xml` or `text/xml` for XML serialization instead of JSON serialization but this has known issues. If the `Content-Type` or `Accept` headers are missing, or they contain only unsupported content types, `application/json` is used.

Do define the `Content-Type` and `Accept` headers when possible. Future versions of M-Files Web Service might add more content options such as `application/html`, `application/xml` or `application/x-www-form-urlencoded`. Currently passing `application/html` as the `Accept` header results in JSON serialization but if M-Files Web Service starts supporting `application/html` this will change and requires modifications to the application.

5. Resources

The resources make up most of the M-Files Web Service interface. These resources allow users to request and modify information stored within M-Files. See 2 for more information on how to use these resources.

Each web service resource has its own URI or URI pattern by which it can be referenced in the HTTP requests. These patterns contain placeholder tokens which can be substituted for different values when making the request. These tokens are listed below in table 5.1.

Table 5.1: *MFWS URI tokens*

Token	*Regex expansion*	*Meaning*
`(type)`	`(\d+)`	Object type of an object.
`(objectid)`	`(\d+|e[^/])`	ID of an object or value list item. External IDs are prefixed with 'e'.
`(version)`	`(\d+|latest|)`	Object version. Optional, will refer to the latest if omitted.
`(file)`	`(\d+)`	Object file ID.
`(id)`	`(\d+)`	A numerical ID used with different structures.
`(path)`	`([^/]+/)*`	View path. May be completely empty. See 7.2 for encoding reference.

5.1 Objects

5.1.1 Objects

`/objects`

Collection of objects in the document vault.

GET

`Results<ObjectVersion>`

Retrieves objects. The amount of returned objects is limited by the server. See search encoding for how to further specify the objects.

Notes

Earlier 9.0.3372.x M-Files versions have an issue preventing the use of this resource. The search within root view (/views/objects) resource is identical to this one and can be used as a working alternative.

5.1.2 Objects of type

`/objects/(type)`

Collection of objects filtered by object type.

GET

`Results<ObjectVersion>`

Retrieves objects of the given type. The amount of returned objects is limited by the server. See /objects for how to further specify the objects.

POST

`ObjectCreationInfo`

Returns

`ObjectVersion`

Creates a new object.

5.1.3 Object

`/objects/(type)/(objectid)`

A single object.

5.1.4 Object version

`/objects/(type)/(objectid)/(version)`

A single object version.

GET	`ExtendedObjectVersion` Retrieves the object information.
?include	A list of additional fields to include in the `ExtendedObjectVersion`. Currently only 'properties' is supported. 'properties' includes the Properties field in the returned object.

DELETE	`ObjectVersion, HTTP 204` Destroys the object version. As checked in versions cannot be destroyed this can only be performed on a checked out version and is equivalent to an undo checkout. Returns the new `ObjectVersion` information if it is still visible to the user. See /objects/(type)/(objectid)/deleted for marking the object as deleted.
?force	If true, DELETE will perform an undo checkout even if the object isn't checked out to the current user on this computer.
?allVersions	If true, DELETE will destroy the whole object. Unlike normal DELETE, this will not require the object to be checked out. Will return `HTTP 204` on successful destroy.

Object properties

5.1.4.1 Object - Deleted

`/objects/(type)/(objectid)/deleted`

Resource representing the Deleted-state of an object.

GET	`bool` Retrieves the deleted status of the object.

PUT	`bool`
Returns	`ObjectVersion` Sets the deleted status of the object.

Notes	The deleted status is tracked by the `Deleted` property and this resource is provided as a convenient access to that. It is still possible to alter that property directly using /objects/(type)/(objectid)/(version)/properties or similar resource.

5.1.4.2 Object - History

`/objects/(type)/(objectid)/history`

Resource listing the full object version history.

GET	`ObjectVersion[]` Retrieves all the available versions of the object.
Notes	The use of this resource is recommended over just traversing through all the possible versions between 1 and the latest one. Some of these versions might be deleted for various reasons which makes them unavailable. This resource retrieves only the available versions.

5.1.4.3 Object - Comments

`/objects/(type)/(objectid)/(version)/comments`

Resource listing the full object comment history.

GET	`VersionComment[]` Retrieves the comments written on the object.

PUT
Returns
	`string`
	`ExtendedObjectVersion`
	Sets the comment on an object.

Notes	Adding a comment might create a new version of the object if it isn't checked out.

5.1.4.4 Object - Check out status

`/objects/(type)/(objectid)/(version)/checkedout`

Resource representing the check out state of the object.

GET	`MFCheckOutStatus` Retrieves the current check out status.

PUT
Returns
	`MFCheckOutStatus`
	`ObjectVersion`
	Sets the check out status. This is allowed only when the object isn't checked out to someone else, that is when the check out status isn't CheckedOut.

Notes	See DELETE on /objects/(type)/(objectid)/(version) for undoing the check out.

5.1.4.5 Object - Object title

`/objects/(type)/(objectid)/(version)/title`

Resource representing the object title.

GET	`string`
	Retrieves the object name
PUT	`string`
Returns	`ObjectVersion`
	Sets the object name
Notes	If the object has an automatic title PUT will result in `HTTP 401`.

5.1.4.6 Object - Object workflow state

`/objects/(type)/(objectid)/(version)/workflowstate`

Resource representing the object workflow state.

GET	`ObjectWorkflowState`
	Retrieves the current workflow state.
PUT	`ObjectWorkflowState`
Returns	`ExtendedObjectVersion`
	Sets the workflow state.

5.1.4.7 Object - Object thumbnail

`/objects/(type)/(objectid)/(version)/preview`

The object thumbnail.

GET	`Stream (application/png)`
	Retrieves the object preview. If a preview cannot be generated the service responds with a `HTTP 404`.
?force	If true, the server streams an empty image instead of giving a `HTTP 404` response if the preview cannot be generated.
?size	Specifies square dimensions. Defaults to 32. This also works with the force-parameter by forcing the empty image to have the specified dimensions.
?width	Specifies preview width. Overrides size.
?height	Specifies preview height. Overrides size.

5.1.4.8 Object - Relationships

`/objects/(type)/(objectid)/(version)/relationships`

A collection of related objects.

GET	`ObjectVersion[], Lookup[]` Retrieves related objects.
?objtype	Only returns related objects of a certain object type.
?direction	Only returns related objects with the specified direction of relationship. 'from' considers only the relationships specified on the current object. 'to' considers only the relationships originating from the related objects. 'both' is the default behaviour which considers both directions.
?type	Specifies the response type. 'lookup' will make the response to be serialized as Lookup[] while the default 'objectversion' will result in the response being serialized as ObjectVersion[]

5.1.4.9 Object - Relationship count

`/objects/(type)/(objectid)/(version)/relationships/count`

The count of the related objects.

GET	`int` Retrieves the amount of related objects.
?objtype	See /objects/(type)/(objectid)/(version)/relationships
?direction	See /objects/(type)/(objectid)/(version)/relationships
?type	See /objects/(type)/(objectid)/(version)/relationships

5.1.4.10 Object - Sub-objects

`/objects/(type)/(objectid)/(version)/subobjects`

A collection of sub-objects.

GET	`ObjectVersion[]` Retrieves the sub-objects.

5.1.4.11 Object - Sub-object count

`/objects/(type)/(objectid)/(version)/subobjects/count`

The count of the sub-objects.

GET	`int` Retrieves the amount of sub-objects.

5.2 Object properties

5.2.1 Object properties

`/objects/(type)/(objectid)/(version)/properties`

The properties of an object.

GET	`PropertyValue[]`
	Retrieves the object properties
?forDisplay	If true, the response will be filtered and sorted for display. Each object has several built-in properties which usually aren't shown to the user. This parameter can be used to leave those out.

POST	`PropertyValue[]`
Returns	`ExtendedObjectVersion`
	Sets the posted properties on the object. If the object already has a value for the sent properties this value will be overridden.

PUT	`PropertyValue[]`
Returns	`ExtendedObjectVersion`
	Sets the object properties. Properties not included in the request are removed from the object.

5.2.2 Single object property

`/objects/(type)/(objectid)/(version)/properties/(id)`

A single property of an object.

GET	`PropertyValue`
	Retrieves a single property value

PUT	`PropertyValue`
Returns	`ExtendedObjectVersion`
	Sets a single property value

DELETE	`ExtendedObjectVersion`
	Removes a single property value

5.2.3 Properties of multiple objects

`/objects/properties`

A resource that allows access to properties of multiple objects.

GET	`PropertyValue[][]` Retrieves properties of multiple objects. The object versions are specified in the URI seperated with semicolons.
Example	`GET /objects/properties;0/5/6;0/112/latest;136/2/3`
POST	`ObjVer[]`
Returns	`PropertyValue[][]` Retrieves properties of multiple objects. The object versions are specified in the request body.
Notes	The properties returned in the response are arranged in the same order as the object versions sent in the request. This information should be used to match the correct PropertyValue[] to the correct object version.

5.3 Object files

5.3.1 Object files

`/objects/(type)/(objectid)/(version)/files`

Files belonging to an object.

GET	`ObjectFile[]` Retrieves the object file information for all the files on an object.
POST	`Stream (application/octet-stream, multipart/form-data)`
Returns	`ObjectVersion` Adds a new file to the object.

5.3.2 Object file

`/objects/(type)/(objectid)/(version)/files/(file)`

A single file on an object.

GET	`ObjectFile` Retrieves the object file infromation for the object file.
DELETE	`HTTP 203` Removes the file from the object.

Object file properties

5.3.2.1 Object file - Contents

`/objects/(type)/(objectid)/(version)/files/(file)/content`

The contents of a single file.

GET	`Stream (application/octet-stream)`
	Retrieves the object file contents.
?mac	If true, appends a SHA-512 message authentication code at the end of the stream.
X-Hmac	Specifies the HMAC key

PUT	`Stream (application/octet-stream, multipart/form-data )`
Returns	`ObjectVersion`
	Replaces the object file contents.

Notes	Supports Range header on GET:

5.3.2.2 Object file - File thumbnail

`/objects/(type)/(objectid)/(version)/files/(file)/preview`

File thumbnail.

GET	`Stream (application/png)`
	Retrieves the file preview. If a preview cannot be generated the service responds with a `HTTP 404`.
?force	See /objects/(type)/(objectid)/(version)/preview
?size	See /objects/(type)/(objectid)/(version)/preview
?width	See /objects/(type)/(objectid)/(version)/preview
?height	See /objects/(type)/(objectid)/(version)/preview

5.3.2.3 Object file - File name

`/objects/(type)/(objectid)/(version)/files/(file)/title`

The file name of an object file.

GET	`string`
	Retrieves the current object file name.

PUT	`string`
Returns	`ObjectVersion`
	Sets the name on the object file.

5.4 Uploading

5.4.1 Temporary upload

`/files`

A collection of temporary uploads.

POST	`Stream (application/octet-stream, multipart/form-data )`
Returns	`UploadInfo, UploadInfo[]`
	Stores a temporary file on the server and assigns an ID for it.
	Once uploaded the file can be used in object creation.

5.5 User session

5.5.1 Session

`/session`

Current user session information.

GET	`SessionInfo`
	Retrieves the current session information.

PUT	`Authentication`
Returns	`Vault[]`
	Performs login using the credentials in the request.

DELETE	`HTTP 204`
	Performs a logout for the session.

5.5.2 Vault

`/session/vault`

The document vault attached to the current session.

GET	`Vault`
	Retrieves the current vault.

PUT	`Vault`
Returns	`Vault`
	Sets the current vault.
	The request must have either the GUID or the Name of the vault filled. In case both of these are filled the GUID is used. If only the name is filled and there are multiple vaults with the same name, the server will respond with `HTTP 409`.

5.5.3 Authentication token

`/session/authenticationtoken`

Authentication token representing the current session.

GET	`string` Retrieves the authentication token for the current session.

5.6 Server information

5.6.1 Server resources

`/server`

Server-level resources.

5.6.2 Server public key

`/server/publickey`

The server public key used to secure messages between the client and the server.

GET	`PublicKey` Used to encrypt secure information before sending it to the server.

5.6.3 Server status

`/server/status`

Server status.

GET	`StatusResponse` Retrieves the server status. If the M-Files server is unavailable the service responds with `HTTP 503`.

5.6.4 Vaults

`/server/vaults`

Collection of vaults on the server.

GET	`Vault[]` Retrieves the vaults from the server
?online	If true, return only online vaults.

5.6.5 Authentication tokens

`/server/authenticationtokens`

Transient resource used in calculating new authentication tokens.

POST	`Authentication`
Returns	`string` Creates a new authentication token based on the authentication information.

5.7 Vault structure

5.7.1 Vault structure

`/structure`

Vault metadata structure information.

5.7.2 Object types

`/structure/objecttypes`

Collection of object type information.

GET	`ObjType[]` Retrieves information on all object types.
?type	Only returns the object types of specific kind. 'real' returns only real object types. 'valuelist' returns only non-object type valuelists. 'both' is the default which returns all object types.

Notes	Internally M-Files considers both valuelists and the real object types as object types. By default this resource will return both kinds of types. If you want only one kind use the type-parameter.

5.7.3 Object type

`/structure/objecttypes/(type)`

Information on a single object type.

GET	`ObjType` Retrieves information on an object type.

Object type properties

5.7.3.1 Object type - Object type icon

`/structure/objecttypes/(type)/icon`

GET	`Stream (application/png )` Retrieves the object type icon.
?size	Icon dimension. Default is 16.

Notes	The icons supports only certain sizes and the real size on the received icon in the respponse might differ from the specified size.

5.7.3.2 Object type - External source refresh

`/structure/objecttypes/(type)/refreshstatus`

The refresh status for an external object type.

PUT	`MFRefreshStatus`
Returns	`MFRefreshStatus`
	Sets the refresh status to either Full or Quick.

5.7.4 Object classes

`/structure/classes`

Collection of object class information.

GET	`ObjectClass[], ClassGroup[]`
	Retrieves information on all object classes.
?objtype	Object type ID. Filters the returned classes by object type. Only classes belonging to the object type are returned.
?bygroup	If true, returns the object classes in class groups.

5.7.5 Object class

`/structure/classes/(id)`

Information on a single object class.

GET	`ExtendedObjectClass`
	Retrieves information on an object class.
?include	Comma separated list of additional data sets to return.
	Currently only ”templates” is supported which will will cause the response to include a list of available templates available to the class.

Object class properties

5.7.5.1 Object class - Class icon

`/structure/classes/(id)/icon`

GET	`Stream (application/png )`
	Retrieves the object class icon.
?size	Icon dimension. Default is 16.

Notes	The icons supports only certain sizes and the real size on the received icon in the respponse might differ from the specified size.

5.7.6 Property definitions

`/structure/properties`

Collection of property definitions.

GET	`PropertyDef[]` Retrieves information on all property definitions.

5.7.7 Property definition

`/structure/properties/(id)`

Information on a single property definition.

GET	`PropertyDef` Retrieves information on a single property definition.

5.7.8 Workflows

`/structure/workflows`

Collection of workflows.

GET	`Workflow[]` Retrieves information on all workflows.

5.7.9 Workflow

`/structure/workflows/(id)`

Information on a single workflow.

GET	`Workflow` Retrieves information on a single workflow.

5.7.10 Workflow states

`/structure/workflows/(id)/states`

Collection of states under a single workflow.

GET	`WorkflowState[]` Retrieves information on all workflow states of the given workflow.
?currentstate	'null' or state ID. Restricts the list of returned states to those that are available as valid states from the current state.

5.7.11 Workflow state

`/structure/workflows/(id)/states/(id)`

Information on a single workflow state.

GET	`WorkflowState` Retrieves information on the specific workflow state.

5.8 Value Lists

5.8.1 Value list items

`/valuelists/(id)/items`

Collection of value list items for a single value list.

GET	`Results<ValueListItem>`
	Retrieves value list item information.
?propertydef	Filters the items using the filter defined on the property definition.
?pN	Filter items using defined property ID and its value. See search conditions.
?parent	Filters items by parent item.
?owner	Filters items by owner item.
?name	Filter using name. Supports wildcards.
?page	Retrieves only one page. Default page size is 100.
?pagesize	Defines page size. If page size is defined, retrieve only one page, default to first.

POST	`ValueListItem`
Returns	`ValueListItem`
	Creates a new value list item in the value list.

5.8.2 Value list item

`/valuelists/(id)/items/(objectid)`

Single value list item information.

GET	`ValueListItem`
	Retrieves a single value list item information.

DELETE	`ValueListItem`
	Deletes a value list item.

Value list item properties

5.8.2.1 Value list item - Value list item title

`/valuelists/(id)/items/(objectid)/title`

The title of a value list item.

GET	`string`
	Retrieves the value list item title.

PUT	`string`
Returns	`ValueListItem`
	Changes the value list item title.

5.9 Views

5.9.1 View

`/views/(path)`

A single view in the view hierarchy.

5.9.2 View contents

`/views/(path)/items`

Contents of a single view.

GET	`FolderContentItems`
	Retrieves the view contents

5.9.3 View contents count

`/views/(path)/items/count`

The count of items within the single view.

GET	`int`
	Retrieves the amount of items in the view.

5.9.4 Search view contents

`/views/(path)/objects`

All objects found within the view.

GET	`Results<ObjectVersion>`
	Searches for objects within the view

5.9.5 Favorites

`/favorites`

Collection of favorited objects.

GET	`ObjectVersion[]`
	Retrieves favorite objects.

POST	`ObjID`
Returns	`ExtendedObjectVersion`
	Adds an object to the favorites.

5.9.6 Favorite object

`/favorites/(type)/(objectid)`

A single favorite object.

GET	`ObjectVersion` Retrieves object version information on the favorite object.
DELETE	`ExtendedObjectVersion` Removes an object from favorites.

5.9.7 Recently accessed by the user

`/recentlyaccessedbyme`

A collection of objects recently accessed by the current user.

GET	`ObjectVersion[]` Retrieves the recently accessed objects.
POST	`ObjID`
Returns	`ExtendedObjectVersion` Notifies object access and adds the object to the recently accessed objects.

6. Resource types

The request and response formats of resources are defined in terms of common types. Most of the types derive directly from those of M-Files API while some of the types are unique to M-Files Web Service.

6.1 JSON data types

When the object instances of these types are transmitted over HTTP protocol they are serialized using one of the supported serialization formats, which currently include only JSON currently. Since M-Files Web Service uses richer type system than offered by JSON format, some of the types are represented using other types. The JSON format used by the types in M-Files Web Service are described in table 6.1 below.

Table 6.1: *Data types of the resource type properties.*

Type	*JSON serialization format*
`string`	JSON string.
`int`	JSON number.
`double`	JSON number.
`bool`	JSON boolean.
`DateTime`	Converted to ISO-8601 format and serialized as string.
Arrays	JSON array.
Other objects	JSON object.
Enumerations	JSON number.

6.2 Classes

6.3 ObjectCreationInfo

Specifies the information required when creating a new object.

```
PropertyValues                    :  PropertyValue[]
Files                             :  UploadInfo[]
```

6.4 UploadInfo

Contains the information on a temporary upload.

```
UploadID                          :  int
Title                             :  string
Extension                         :  string
Size                              :  long
```

6.5 TypedValue

A 'typed value' represents a value, such as text, number, date or lookup item.

DataType	:	MFDataType
HasValue	:	bool
Value	:	object
Lookup	:	Lookup
Lookups	:	Lookup[]
DisplayValue	:	string
SortingKey	:	string
SerializedValue	:	string

6.6 ExtendedObjectVersion

An object version with extended properties. Inherits from ObjectVersion.

Properties	:	PropertyValue []

6.7 Authentication

Authentication details.

Username	:	string
Password	:	string
Domain	:	string
WindowsUser	:	bool
ComputerName	:	string
VaultGuid	:	string
Expiration	:	DateTime?
ReadOnly	:	bool
URL	:	string
Method	:	string

6.8 Vault

Vault information.

Name	:	string
GUID	:	string
Authentication	:	string

6.9 PasswordChange

Information required for changing password.

OldPassword	:	string
NewPassword	:	string

6.10 PublicKey

Server public key information.

Exponent	:	string
Modulus	:	string

6.11 StatusResponse

Response for status requests.

Successful	:	bool
Message	:	string

6.12 ExtendedObjectClass

An object class with extended properties. Inherits from ObjectClass.

AssociatedPropertyDefs	:	AssociatedPropertyDef[]
Templates	:	ObjectVersion[]

6.13 WorkflowState

Workflow state information.

Name	:	string
ID	:	int
Selectable	:	bool

6.14 FolderContentItems

An object version with extended properties. Inherits from ObjectVersion.

Path	:	string
MoreResults	:	bool
Items	:	FolderContentItem[]

6.15 ObjectWorkflowState

A workflow state on an object.

State	:	PropertyValue
StateID	:	int
StateName	:	string
Workflow	:	PropertyValue
WorkflowID	:	int
WorkflowName	:	string
VersionComment	:	string

6.16 WebServiceError

M-Files Web Service error object.

Status	:	int
URL	:	string
Method	:	string
Exception	:	ExceptionInfo

6.17 ExceptionInfo

Message	:	string
InnerException	:	ExceptionInfo
Stack	:	StackTraceElement[]

6.18 StackTraceElement

M-Files Web Service error stack trace element.

FileName	:	string
LineNumber	:	int
ClassName	:	string
MethodName	:	string

6.19 Results¡T¿

Results of a query which might leave only a partial set of items.

Items	:	T[]
MoreResults	:	bool

6.20 PrimitiveType¡T¿

Value	:	T

6.21 ObjectVersion

Based on M-Files API.

AccessedByMeUtc	:	DateTime
CheckedOutAtUtc	:	DateTime
CheckedOutTo	:	int
CheckedOutToUserName	:	string
Class	:	int
CreatedUtc	:	DateTime
Deleted	:	bool
DisplayID	:	string
Files	:	ObjectFile []
HasAssignments	:	bool
HasRelationshipsFromThis	:	bool
HasRelationshipsToThis	:	bool
IsStub	:	bool
LastModifiedUtc	:	DateTime
ObjectCheckedOut	:	bool
ObjectCheckedOutToThisUser	:	bool
ObjectVersionFlags	:	MFObjectVersionFlag
ObjVer	:	ObjVer
SingleFile	:	bool
ThisVersionLatestToThisUser	:	bool
Title	:	string
VisibleAfterOperation	:	bool

6.22 ObjectFile

Based on M-Files API.

ChangeTimeUtc	:	DateTime
Extension	:	string
ID	:	int
Name	:	string
Version	:	int

6.23 ObjVer

Based on M-Files API.

ID	:	int
Type	:	int
Version	:	int

6.24 PropertyValue

Based on M-Files API.

```
PropertyDef                         :  int
TypedValue                          :  TypedValue
```

6.25 SessionInfo

Based on M-Files API.

```
AccountName                         :  string
ACLMode                             :  MFACLMode
AuthenticationType                  :  MFAuthType
CanForceUndoCheckout                :  bool
CanManageCommonUISettings           :  bool
CanManageCommonViews                :  bool
CanManageTraditionalFolders         :  bool
CanMaterializeViews                 :  bool
CanSeeAllObjects                    :  bool
CanSeeDeletedObjects                :  bool
InternalUser                        :  bool
LicenseAllowsModifications          :  bool
UserID                              :  int
```

6.26 ObjType

Based on M-Files API.

```
AllowAdding                         :  bool
CanHaveFiles                        :  bool
DefaultPropertyDef                  :  int
External                            :  bool
ID                                  :  int
NamePlural                          :  string
Name                                :  string
OwnerPropertyDef                    :  int
ReadOnlyPropertiesDuringInsert      :  int []
ReadOnlyPropertiesDuringUpdate      :  int []
RealObjectType                      :  bool
```

6.27 PropertyDef

Based on M-Files API.

```
AllObjectTypes                  :  bool
AutomaticValue                  :  string
AutomaticValueType              :  MFAutomaticValueType
BasedOnValueList                :  bool
DataType                        :  MFDataType
ID                              :  int
Name                            :  string
ObjectType                      :  int
ValueList                       :  int
```

6.28 Workflow

Based on M-Files API.

```
ID                              :  int
Name                            :  string
ObjectClass                     :  int
```

6.29 ValueListItem

Based on M-Files API.

```
DisplayID                       :  string
HasOwner                        :  bool
HasParent                       :  bool
ID                              :  int
Name                            :  string
OwnerID                         :  int
ParentID                        :  int
ValueListID                     :  int
```

6.30 ObjID

Based on M-Files API.

```
ID                              :  int
Type                            :  int
```

6.31 Lookup

Based on M-Files API.

```
Deleted                         :  bool
DisplayValue                    :  string
Hidden                          :  bool
Item                            :  int
Version                         :  int
```

6.32 VersionComment

Based on M-Files API.

LastModifiedBy	:	PropertyValue
ObjVer	:	ObjVer
StatusChanged	:	PropertyValue
Comment	:	PropertyValue

6.33 AssociatedPropertyDef

Based on M-Files API.

PropertyDef	:	int
Required	:	bool

6.34 FolderContentItem

Based on M-Files API.

FolderContentItemType	:	MFFolderContentItemType
ObjectVersion	:	ObjectVersion
PropertyFolder	:	TypedValue
TraditionalFolder	:	Lookup
View	:	View

6.35 View

Based on M-Files API.

Common	:	bool
ID	:	int
Name	:	string
Parent	:	int
ViewLocation	:	ViewLocation

6.36 ViewLocation

Based on M-Files API.

OverlappedFolder	:	TypedValue
Overlapping	:	bool

6.37 ObjectClass

Based on M-Files API.

ID	:	int
Name	:	string
NamePropertyDef	:	int
Workflow	:	int

6.38 ClassGroup

Based on M-Files API.

```
Name                                    :  string
```

6.39 Enumerations

6.40 MFObjectVersionFlag

```
0          :  None
1          :  Completed
2          :  HasRelatedObjects
```

6.41 MFAuthType

```
0          :  Unknown
1          :  LoggedOnWindowsUser
2          :  SpecificWindowsUser
3          :  SpecificMFilesUser
```

6.42 MFACLMode

```
0          :  Simple
1          :  AutomaticPermissionsWithComponents
```

6.43 MFDataType

```
0          :  Uninitialized
1          :  Text
2          :  Integer
3          :  Floating
5          :  Date
6          :  Time
7          :  Timestamp
8          :  Boolean
9          :  Lookup
10         :  MultiSelectLookup
11         :  Integer64
12         :  FILETIME
13         :  MultiLineText
14         :  ACL
```

6.44 MFAutomaticValueType

0	:	None
1	:	CalculatedWithPlaceholders
2	:	CalculatedWithVBScript
3	:	AutoNumberSimple
4	:	WithVBScript

6.45 MFFolderContentItemType

0	:	Unknown
1	:	ViewFolder
2	:	PropertyFolder
3	:	TraditionalFolder
4	:	ObjectVersion

7. Syntax encoding

7.1 Typed values

The value part of the typed value is encoded according to the data type. The encoding does not include the data type as this is often defined by the situation. If this is not the case the data type must be communicated together with the encoded value. Table 7.1 lists the different encoding schemes for different data types.

Table 7.1: *Typed value encoding by data type.*

Data type	*Encoding*
ACL	Not implemented.
Boolean	'true' / 'false'
Time, Date, Timestamp	ISO format string.
Floating	Real number as string
Integer, Integer64	Integer as string
Lookup	Integer as string. Lookup ID.
Multi-Select Lookup	Comma separated Lookup-encoding.
Uninitialized	Not implemented.
Text, Multi-Line Text	string

7.2 Views

View paths are encoded in the URL path based on the level hierarchy. Levels refer to views, traditional folders or virtual folders defined with typed values. The encoding format is described below in table 7.2. Example 7.1 contains encoding samples for different views.

Table 7.2: *ABNF of the view path encoding.*

```
viewpath        = '/' / level *level [ '/' ]
level           = '/' ( view-level / folder-level / typed-value )
view-level      = 'v' number ; Normal view
folder-level    = 'y' number ; Traditional folder
typed-value     = data-type data-value
data-type       = 'T' / 'M' ; Text, MultiLineText
                / 'I' / 'J' / 'R' ; Integer, Integer64, Real number
                / 'D' / 'C' / 'E' / 'P' ; Date, Time, FILETIME, Timestamp
                / 'L' / 'S' ; Lookup, MSLU
                / '-' / 'A' / 'B' ; Uninitialized, ACL, Boolean
data-value      = segment-nz
                ; URI path segment as defined in RFC-3986.  (ie. legal URI text,
                excluding '/')
                ; The value is encoded according to 5.1 and then double URI encoded.
```

```
1
2  // Root: views/items
3  $.get( "http://example.org/REST/views/items.aspx", callback, "json" );
4
5  // "1. Documents" view of Sample vault.
6  $.get( "http://example.org/REST/views/V133/items.aspx", callback, "json" );
7
8  // "1. Documents\By Customer\ESTT Corporation (IT)" of Sample vault.
9  $.get( "http://example.org/REST/views/V133/V136/L141/items.aspx", callback,
       "json" );
10
11 // "By Keyword\MFWA / MFWS" view (View ID 123)
12 $.get( "http://example.org/REST/views/V123/TMFWA&2520%252F%2520MFWS/items.aspx",
       callback, "json" );
13 // or
14 $.get( "http://example.org/REST/views/V123/TMFWA %252F MFWS/items.aspx",
       callback, "json" );
```

Listing 7.1: *View encoding examples*

The double uri encoding is not strictly required in most cases. If the text value passed in the URI contains a forward slash ('/') then the double encoding must be used. Without double encoding the forward slash will be interpreted as a view level separator.

7.3 Search conditions

Search conditions are conveyed as query parameters. The parameter starts with an expression specifier. This is followed by the operator followed by the value. The condition encoding is described in table 7.3. Example 7.2 contains some encoding samples for different searches.

Table 7.3: *ABNF of the search condition encoding.*

```
conditions      = condition *( '&' condition )
condition       = expression operator value
expression      = 'q' ; Quicksearch
                / 'p' number ; PropertyValueExpression
                / 'vl' number ; TypedValueExpression (Value list search)
                / 'o' ; Object Type
                / 'd' ; Is Deleted
operator        = [ '!'  ]  op-mod
op-mod          = '=' ; Equal
                / '<<=' / '<=' ; Less, Less or Equal
                / '>>=' / '>=' ; Greater, Greater or Equal
                / '**=' / '*=' ; Matches wildcard, Contains
                / '^' ; Starts With
value           = null-value / id-value / typed-value-encoded-value / 'include'
                ; See 5.1 for the typed value encoding.
                ; 'include' is a special value available for the 'Is Deleted' condition.
null-value      = '%00' ; Equals null
id-value        = number / 'e' string
                ; 'e' specifies external ID condition.  ID is URI escaped.
```

```
1
2
3  // Quick search for "specification"
4  $.get( "http://example.org/REST/objects.aspx?q=specification", callback, "json"
       );
5
6  // Quick search for "web service"
7  $.get( "http://example.org/REST/objects.aspx?q=web service", callback, "json" );
8
9  // Objects where the title contains "m-files"
10 $.get( "http://example.org/REST/objects.aspx?p0*=m-files", callback, "json" );
11
12 // Deleted documents (Object type ID 0).
13 $.get( "http://example.org/REST/objects.aspx?d=true&o=0", callback, "json" );
14
15 // Quick search for "web service". Also include deleted objects.
16 $.get( "http://example.org/REST/objects.aspx?q=web service&d=include", callback,
       "json" );
17
18 // Note: Searching by external ID can be done with direct object reference.
19 // Customer with object ID "123A"
20 $.get( "http://example.org/REST/objects/136/e123A.aspx", callback, "json" );
```

Listing 7.2: *Search encoding examples*

Note that unlike M-Files API, M-Files Web Service doesn't return deleted objects by default. Specify `d=include` *condition to include the deleted objects as well.*

8. Error reporting

Inevitably at some point there will be an error during one of the M-Files Web Service requests. When this happens, M-Files Web Service interrupts the request processing and responds with a 4xx or 5xx HTTP status code instead.

The HTTP status codes in the 4xx and 5xx range indicate an error during the request. If the error happened inside M-Files Web Service, the error description is included in the HTTP response body as a `WebServiceError` object. The object contains `Message` property which should provide more information on the error.

The message property in the `WebServiceError` departs from the traditional M-Files error reporting familiar from M-Files API. In M-Files API, especially for argument errors, the range of different error messages is quite small. This makes it possible to localize all the error messages but it also results in less specific messages. M-Files Web Service contains more specific error messages at the cost of localization.

`WebServiceError` also contains the original stack trace from the server. This stack trace is valuable if there is ever a need to contact M-Files support.

The returned JSON also contains `ErrorCode` property. Do not use this for anything. The information in this property is wrong and the property will be changed in future M-Files versions.

9. Compatibility

9.1 Server compatibility

M-Files Web Service requires .NET Framework 4 and IIS 5.1 or newer on the server. There are couple of features which enable better support for older IIS versions but might require changes in the requests.

9.1.1 File extensions

M-Files Web Service requires .NET Framework 4 and IIS 5.1 or newer on the server. Older IIS versions (5.1 and 6.0) use file extensions to map incoming HTTP requests to different extensions such as ASP.Net. M-Files Web Service supports `.aspx` and `.ashx` extensions for all resource URIs. These make it easier to invoke the resources in case the web service is hosted on top of IIS 5.1 or IIS 6.0 server and have no real effect on the actual resource.

When writing an application that might be consuming M-Files Web Service from an older IIS server, do use either `.aspx` or `.ashx` extension in the requests. This makes it easier to configure the server and enables the default M-Files Web Access deployment to be used.

9.1.2 HTTP methods

Another compatibility hurdle caused by the older IIS versions is the support for `PUT` and `DELETE` methods in the HTTP requests. The older IIS versions forward only the `GET` and `POST` requests to ASP.Net by default. M-Files Web Service supports using a `_method` query parameter to override the HTTP method in the request. See table 9.1 below for how to use this parameter.

Table 9.1: *Using `_method` parameter.*

Original request	*Compatible request*
`GET /REST/resource`	`GET /REST/resource`
`POST /REST/resource`	`POST /REST/resource`
`PUT /REST/resource`	`POST /REST/resource?_method=PUT`
`DELETE /REST/resource`	`POST /REST/resource?_method=DELETE`

Use the `_method` parameter to invoke `PUT` and `DELETE` methods over `POST`. This enables the application to consume M-Files Web Services that are hosted on top of an older IIS server.

9.1.3 REST path in a query parameter

Some third party components only support web services with a single URL and insist on passing all parameters as query parameters. M-Files Web Service can be used by such components by hosting it within an `.ashx` handler. The M-Files Web Access includes `MFWS.ashx` handler for such purposes. This handler is able to invoke any M-Files Web Service resource by passing the resource path in the `restPath` query parameter. Table 9.2 below shows M-Files Web Service requests using MFWS.ashx handler.

Table 9.2: *Using MFWS.ashx to call M-Files Web Service.*

Original request	*Compatible request*
`GET /REST/resource`	`GET MFWS.ashx?restPath=resource`
`POST /REST/resource`	`POST MFWS.ashx?restPath=resource`

10. Code samples

10.1 Package

The sample code package for M-Files Web Service can be downloaded from http://www.m-files.com/mfws/MFWS_Samples.zip. The package contains code samples and utilities in different languages. The package structure is described in table 10.1.

Table 10.1: *M-Files Web Service sample package structure*

Directory	*Description*
`MFWS Documentation.pdf`	This document.
`/CSharp/Samples`	C♯ samples used in this document.
`/CSharp/Utilities`	C♯ helpers.
`/Java/Utilities`	Java helpers.
`/JavaScript/Samples`	JavaScript samples used in this document.
`/JavaScript/Utilities`	JavaScript helpers.

10.2 M-Files Web Service C♯ Client

The M-Files Web Service client contained in the C♯ utilities enables quick access to M-Files Web Service. The client takes care of most of the compatibility issues, authentication, serialization and error handling. Example 10.1 below demonstrates how to use the client to read and edit M-Files contents.

```
1
2
3  static void Run()
4  {
5      // Create the MFWSClient.
6      MfwsClient client = new MfwsClient("http://example.org/REST");
7
8      // Acquire an authentication token.
9      // This involves performing a POST request to /server/authenticationtokens
10     // resource. The return value is PrimitiveType<string> which contains the
11     // actual token in the Value property.
12     var result = client.Post<PrimitiveType<string>>(
13         "/server/authenticationtokens",
14         new Authentication { Username = "username", Password = "password" });
15
16     // Read the authentication token and store it in the client.
17     // This way it will be used in future requests.
18     client.Authentication = result.Value;
19
20     // The current authentication token is an application-level token as we
21     // didn't specify a vault GUID when we constructed it. We cannot use it
22     // to access the Vault. However we can use it to acquire the vault-level
23     // token through the /session/vaults collection.
24
25     // Request the current vaults.
26     var vaults = client.Get<Vault[]>("/session/vaults");
27
28     // Select the first available vault.
29     var vault = vaults[0];
30
31     // The vaults return vault-level authentication tokens. Get the one on
32     // the selected vault and insert it as the default token to the client.
33     client.Authentication = vault.Authentication;
```

```
34
35      // Now we have a client with vault-access. We can now access the vault
36      // contents.
37
38      // Retrieve an object version.
39      ObjectVersion ov = client.Get<ObjectVersion>( "/objects/0/136/latest" );
40      Console.WriteLine( ov.Title );
41
42      // Perform a check out.
43      ov = client.Put<ObjectVersion>(
44          "/objects/0/136/latest/checkedout",
45          new PrimitiveType<MFCheckOutStatus> { Value =
                MFCheckOutStatus.CheckedOutToMe } );
46
47      Console.WriteLine( "Checked out: " + ov.ObjectCheckedOut );
48
49      // Set the name property.
50      ov = client.Put<ObjectVersion>(
51          "/objects/0/136/" + ov.ObjVer.Version + "/properties/0",
52          new PropertyValue
53          {
54              // PropertyDef 0 is the built-in Title property.
55              PropertyDef = 0,
56              TypedValue = new TypedValue
57              {
58                  DataType = MFDataType.Text,
59                  Value = "New Name"
60              }
61          } );
62
63      // Print the new title.
64      Console.WriteLine( ov.Title );
65
66      // Finally check the object in.
67      ov = client.Put<ObjectVersion>(
68          "/objects/0/136/" + ov.ObjVer.Version + "/checkedout",
69          new PrimitiveType<MFCheckOutStatus> { Value = MFCheckOutStatus.CheckedIn
                } );
70
71      Console.WriteLine( "Checked out: " + ov.ObjectCheckedOut );
72
73  }
```

Listing 10.1: *Using M-Files Web Service sample client.*