Sitecore Web API 1.0 User Guide Draft

The Web API Hook

A web request to the Sitecore Web API must start with the special hook in the following format:

*/-/webapi/v<version>*

where *version* is a version number of the web service.

Current version of the Web API is 1, therefore at present the only valid hook is */-/webapi/v1*.

HTTP Parameters

These are the HTTP query string arguments you can use to specify your GET request:

* Fields

Specifies the fields which will be added to the result set. If not specified, all the fields will be added.

Examples:

*fields=Text*

*fields=Title|Text*

Order of the specified fields is not respected: fields in the result set will follow each other alphabetically.

* Payload

Specifies that *lite* result set must be used.

Example: *payload=1*

*1* is only valid value for the parameter. All other values are ignored.

* Scope

Specifies the set of the items we working with.

Possible values are: *s* (self), *p* (parent), and *c* (children). By default, *self* is used.

Examples:

*scope=p*

*scope=c|p*

*scope=s|p|c*

Notice that order of the values is respected: the result set will reflect the specified order.

Using Sitecore Queries

It is possible to use Sitecore queries in Web API requests. Just specify a query through the *query* HTTP parameter. Syntax is the same as used in the Sitecore API: the Web API simply transfers a value of the *query* parameter directly to the corresponding part of the Sitecore API.

Here are few examples how to specify a required scope via Sitecore (Fast) Query:

*?query=/Sitecore/Content/\** - a *regular* Sitecore query is used.

*?query=fast:/Sitecore/Content/\** - a *fast* Sitecore query is used.

**Important note:** Specified Sitecore query affects working items. If specified, working items are items returned by the query; otherwise, working item is item resolved by the URL. After working items are resolved, the *scope* value (if specified) will be applied to each working item.

Information about how to use the Sitecore Fast Query available here:

<http://sdn.sitecore.net/upload/sdn5/developer/using%20sitecore%20fast%20query/using%20sitecore%20fast%20query001.pdf>

Paging

There is a way to get a part of a result set of a GET request using paging. To do that you need to specify page number and page size as described in the following example:

*?page=0&pageSize=10* – returns 1st page containing 10 (or less) items.

*?page=5&pageSize=5* – returns 6th page containing 5 (or less) items.

* page – an integer value which is greater than or equal to 0.
* pageSize – an integer value which is strictly greater than 0.

Creating new items

The “POST” HTTP verb is used to for ‘create’ operation. In order to create a new item one should specify an item name in ‘name’ query string parameter and item template (or branch) in ‘template’ query string parameter. The ‘template’ parameter can be either template ID or relative template path (e.g. ‘Sample/MyTemplate’). When creating items from a branch only ID is supported.

Sample (creating ‘myItem’ item under ‘/sitecore/content/home’ from ‘Sample item’ template in Master database):

<http://localhost/-/webapi/v1/sitecore/shell/sitecore/content/home?name=myItem&template=Sample%2FSample%20Item&sc_database=master>

The response contains the JSON (by default) representation of the newly created item. The response format is identical to ‘Read’ requests and can be also controlled by with fields/payload parameters.

If there are several items resolved as a request scope (i.e. when using query or scope parameter), new item will be created as a child of the first item in a scope. Since the order may be unpredictable it is highly recommended to have a scope of the only item for create operation.

In order to create new item and update its fields at once, one can specify <field\_name> =<field\_value> pairs (please refer ‘Update’ operation for details) in the request body when creating new items.

**Note: Due to Sitecore bug, WebDAV should be turned off (see ‘WebDAV.Enabled’ setting in Sitecore.WebDAV.config) to make ‘Update’ and ‘Delete’ operations work.**

Updating existent items

The PUT HTTP request is used for the UPDATE operation. The scope is the items that will be updated. To update these, the field data must be specified *in the request body* in the traditional format:

*<fieldName1>=<fieldValue1>&<fieldName2>=<fieldValue2> ... &<fieldNameN>=<fieldValueN>*

Example (in a PUT request body):

*firstName=Duane&lastName=Allman*

If an item contains a field with the specified name, the field will be updated with the specified value; otherwise, it will be ignored. Notice that field names are case sensitive.

The UPDATE operation returns a collection of the updated items.

Deleting existent items

The DELETE HTTP request is used for the DELETE operation. The scope is the items that will be deleted. It does nothing for non-existing items. The operation returns a count of the deleted items.

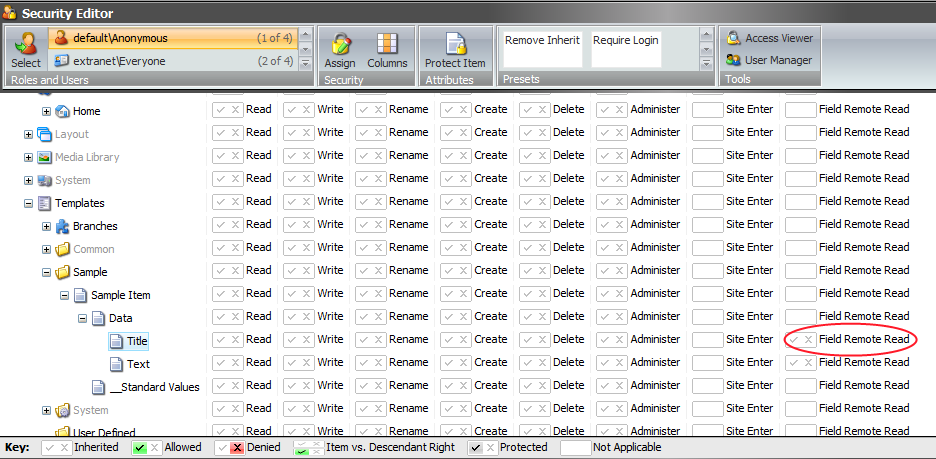
# Sitecore Web API security model

Sitecore Web API appreciates default Sitecore security model.

The default security settings are applied when executing Web API request.

The security model is slightly extended for Web API requests:

1. Field-level ‘Field Remote Read’ access right was introduced. In contrast with ‘Field Read’ access right, the ‘Field Remote Read’ access right is resolved as ‘Allowed’ if it is explicitly allowed for a specific field. By default, it is denied.



Web API client, who was granted ‘Read’ access to specific item, obtains item common properties (Template name, ID, Display Name etc.) and a set of fields, he was granted the ‘ Field Remote Read’ access to.

A client, who has no ‘Read’ access to specific item, obtains no information about it.

1. If a Web API request is targeting a site within Sitecore security domain (e.g. SHELL), the client should be logged in the same domain. Otherwise, the Web API responses with a ‘Request forbidden’ (403) status.

Note: The Web API also responses with 403 status in case of:

- Anonymous client request to a site which requires login

- ‘Site Enter’ access right is not granted for a client

# User authentication

User credentials should be passed with a request using HTTP headers:

* ‘X-Scwebapi-Username ‘ for user name
* ‘X-Scwebapi-Password’ for user password

The authentication is stateless, i.e. user credentials should be passed with each request.

One can be authenticated in WebApi using the authentication cookies. This scenario is useful when using the API in Sitecore backend (when user already has authentication cookies). However, the WebApi doesn’t expose a method for logging in and receiving the authentication cookies.

In order to enable cookies authentication, If credentials shouldn’t be passed in HTTP headers.

## Credentials encrypting

The Sitecore WebAPI supports credentials encryption from the box.

In order to use encrypted the credentials one should:

1. Get the WebAPI encryption public key via the **http://<host\_name>/-/webapi/v1/-/system/securekey** \*(the key is sent in [XML DSig](http://www.w3.org/TR/xmldsig-core/#sec-RSAKeyValue) format as a plain text)
2. Encrypt the credentials with a public key using RSA algorithm
3. Add ‘X-Scwebapi-Encrypted’ header equals ‘1’ to a request

\* The WebAPI requests to resources containing ‘/-/system/’ in their path, have specific semantics and are not supposed for content manipulations (items CRUD operations). These requests have different response format from regular WebAPI requests. The response format and content type may vary depending on the requested operation. Currently only ‘securekey’ operation is supported. Some additional operations (e.g. render rendering) are supposed to appear soon.

Note1: If SSL connection is used for communication, the credentials shouldn’t be encrypted. The server takes care of headers encryption. It is recommended to use SSL connection instead of custom ecryption.

Note2: By default WebAPI server uses 1024-bit encryption key. The value can also be specified as a ‘keyLength’ setting of the RSA encryption provider in .config file.

Note3: The secret encryption key is stored in machine key storage. The default key container name is ‘SCWEBAPIKEYCONTAINER’ (can be changed using ‘keyContainer’ property of the RSA encryption provider)