Sitecore CMS 6.5-6.6

Sitecore Item Web API 1.0.0 Developer's Guide

A developer's guide to the Item Web API.

Table of Contents

The Item Web API Hook

Developers use the Sitecore Item Web API to manipulate the content items in Sitecore through HTTP requests. The API gives access to content through items paths, IDs, and Sitecore queries. This service produces output in JSON format and is both highly customizable and optimized to support a minimum number of requests.

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

`/-/item/v<version>` where `<version>` is a number of the service version.

The current version of the Item Web API is 1, therefore the only valid hook is `/-/item/v1`.

Query String Parameters

The following sections describe the query string parameters that you can use in an Item Web API request:

Item Id

The `sc_itemid` parameter specifies the ID of the content item to be resolved by the Item Web API request.

Example:

```
http://<host_name>/-/item/v1/?sc_itemid={A60ACD61-A6DB-4182-8329-C957982CEC74}
```

Item Version

The `sc_itemversion` parameter specifies the version number of the content item to be resolved by the Item Web API request.

Example:

```
http://<host_name>/-/item/v1/?sc_itemid={A60ACD61-A6DB-4182-8329-C957982CEC74}&sc_itemversion=2
```

If you do not specify the `sc_itemversion` parameter or the specified version does not exist, the latest version of the item is retrieved.

You should not use the `sc_itemversion` parameter:

- If you want a set of items to be resolved by a query in the Item Web API request.
- To add or delete a particular version or an item.

Database

The `sc_database` parameter specifies the database that contains the content items that you want to manipulate.

Example:

```
http://<host_name>/-/item/v1/?sc_itemid={A60ACD61-A6DB-4182-8329-C957982CEC74}&sc_database=master
```

If you do not specify the `sc_database` parameter, the current context database is used.

Note
Only members of the **Sitecore Client Users** role can switch databases.

Language

The `language` parameter specifies the context language for the Item Web API request.

Example:

```
http://<host_name>/-/item/v1/?sc_itemid={A60ACD61-A6DB-4182-8329-C957982CEC74}&language=da-DK
```

The Sitecore default language parameter `sc_lang` stores the language value in a cookie. However, the `language` parameter is stateless. This means that the `language` parameter only sets the language for the current request and subsequent requests are not affected.

If the `language` parameter is not specified, the language cookie, that might have been previously set, has an effect.

To use the default language and ensure that the language specified in the cookie is not used, specify it as `?language=default`.

You should not use the `language` parameter to add or delete the language versions of an item. This functionality does not exist in the Item Web API.

Fields

The `fields` parameter specifies the fields that are added to the result set. If it is not specified, the content fields are added. The field can be specified either by the field name or by the field ID.

Note
The field names and IDs are not case sensitive.

Examples:

```
fields=Text
fields=tExT
fields={A60ACD61-A6DB-4182-8329-C957982CEC74}
fields=Title|Text
fields=tItLe|TeXt
fields=Title|{A60ACD61-A6DB-4182-8329-C957982CEC74}
fields={75577384-3C97-45DA-A847-81B00500E250}|Text
fields={75577384-3C97-45DA-A847-81B00500E250}|{A60ACD61-A6DB-4182-8329-C957982CEC74}
```

The field values in the response are raw values. The raw values are in plain text format and are taken from the corresponding Sitecore database.

To retrieve data from a Binary Large Object (BLOB) field, you should add the `extractblob=1` query parameter to the Item Web API request. The BLOB data in the response is Base64 encoded.

Payload

You can use the `payload` parameter to change the set of item fields in the Item Web API response. An item can contain a lot of fields. Even if its template does not contain any fields, the item still contains the standard fields. If you do not need to retrieve all the item fields, you can reduce the network traffic by just retrieving the item fields that you need, such as **ID**, **Name**, **DisplayName**, **FullPath**, and **TemplateName**.

The `payload` parameter accepts the following values:

- *min* — no fields are returned in the service response.
- *content* — only content fields are returned in the service response.
- *full* — all the item fields, including content and standard fields, are returned in the service response.

Examples:

```
payload=min
payload=content
payload=full
```

Note
The payload parameter works when no fields were specified. If fields are explicitly specified with the `fields` parameter, the `payload` parameter is ignored.

Scope

The `scope` parameter specifies the set of items that you are working with.

You can use the following values to define the scope:

- *s* for self — default if nothing is specified
- *p* for parent
- *c* for children

Examples:

```
scope=p
scope=c|p
scope=s|p|c
```

Note
The order of the values is important because the result set reflects the specified order.

Query

To resolve items, you can use Sitecore queries as HTTP `query` parameters in the Item Web API requests. The syntax of the query is the same as the one that is used in the Sitecore API. The Item Web API transfers the value of the `query` parameter directly to the corresponding part of the Sitecore API.

The following examples show how to specify the required scope in Sitecore Query and in Sitecore Fast Query:

- `query=/Sitecore/Content/*` – a regular Sitecore query.
- `query=fast:/Sitecore/Content/*` – a Sitecore fast query.

For more information about how to use Sitecore Fast Query, see the manual *Using Sitecore Fast Query*.

The item set that is resolved by the query is affected by the `scope` parameter.

Example:

```
http://<host_name>/-/item/v1/?scope=c&query=/sitecore/content/home/*
```

In this example, the query resolves the immediate children of the *Home* item. Since the value of the `scope` parameter is `c`, which stands for the children of the resolved items, only the *Home* item's grandchildren are also included in the request output.

Note
If you want to use Sitecore query with special parameters, for example:

```
query=/sitecore/content/home//*[@@templatekey='sample item']
```

you must encode this query first. To encode a query, use the URL Decoder / Encoder online tool that you can find at http://meyerweb.com/eric/tools/dencoder/.

Note
When you submit a field value as a query parameter, remember that field values are case-sensitive, while item and field names are not. For example:

`/sitecore/content/Home/*[@Title = 'Welcome to Sitecore']` and
`/sitecore/content/Home/*[@Title = 'WELCOME TO SITECORE']` queries are not equivalent.

`/sitecore/content/Home/*[@Title = 'Welcome to Sitecore']` and
`/sitecore/content/Home/*[@TITLE = 'Welcome to Sitecore']` queries are equivalent.

Page

If the response contains numerous result items, you can use paging to obtain parts of the whole result set as pages. You can iterate through the whole result set by pages. A page has a number and a size. The number is the index of a given page in a set that starts with 0. The page size is the number of the result items that are presented in the page.

To retrieve a specific page, you must specify the following HTTP parameters:

- `page` — an integer value which is greater than or equal to `0`.
- `pageSize` — an integer value which is greater than `0`.

Examples:

- `page=0&pageSize=10` — returns the first page that contains 10 or less items.
- `page=5&pageSize=5` — returns the sixth page that contains 5 or less items.

Item Manipulation

The following sections describe how to use Item Web API requests to retrieve, create, update, and delete items.

Retrieving Existing Items

To retrieve existing items, you can use the HTTP `GET` method in your Item Web API request.

To retrieve items, you can use any of the following options:

- Specifying the item ID:

  `http://<host_name>/-/item/v1/?sc_itemid={A60ACD61-A6DB-4182-8329-C957982CEC74}`

- Specifying the query:

  `http://<host_name>/-/item/v1/?query=/sitecore/content/*`

- Specifying the item path:

  `http://<host_name>/-/item/v1/sitecore/content/home`

When you specify the path to the item, remember that the Item Web API supports Sitecore's site resolving mechanism. For example, the `http://<host_name>/-/item/v1/sitecore/shell` request retrieves the start item of the managed *shell* site, not the `/sitecore/shell` item of the managed *website*.

Creating Items

To create an item, you can use the HTTP `POST` method in an Item Web API request.

The required query string parameters are:

- `template` — specifies the template that the new item is based on.

  This parameter accepts the following values:

  - The template ID
  - The relative template path, for example, `Sample/MyTemplate`
  - The branch ID

- `name` — specifies the name of the item being created.

It must be a valid Sitecore item name.

Creating an Item from a Template

To create an item called `MyItem` that is based on the *Sample Item* template in the *master* database, use a URL in the following format:

```
http://<host_name>/-
/item/v1/sitecore/Content/Home?name=MyItem&template=Sample/Sample
Item&sc_database=master
```

Creating Items from a Branch

If you create an item from a branch, you must pass the branch ID and the item name. You cannot pass a relative path instead of the branch ID.

A branch consists of a branch item and its subitems. All of these items are created when you pass the branch ID. The item name you that you submit in a query is used as the branch item name.

Specifying the Insertion Point for a New Item

You can also use the `sc_itemid` or the `query` parameter to specify the parent of the item that you are creating.

If the resolved scope contains more than one item, for example, in the Sitecore query or in the `scope` HTTP parameter, the new item is created as a child of the first item in the scope.

Since the order of the items in the scope is unpredictable, we recommend that the scope only contains the item for the creation operation. This means that you should avoid situations when there is more than one item resolved for the `Create` operation.

Example:

```
http://<host_name>/-/item/v1/?query= /sitecore/Content/Home/*&
name=MyItem&template=Sample/Sample Item&sc_database=master
```

In this example, all the descendants of the *Home* item are resolved and it is hard to know under which of the descendants the new item is created.

Writing the Content to a New Item

You can create a new item and update its fields at the same time by specifying the field values in the POST request body in the following formats:

```
<fieldName1>=<fieldValue1>&<fieldName2>=<fieldValue2>&<fieldNameN>=<fieldValu
eN>
```

or

```
<fieldID1>=<fieldValue1>&<fieldID2>=<fieldValue2>&<fieldIDN>=<fieldValueN>
```

To update the item fields, you must add the `Content-Type=application/x-www-form-urlencoded` HTTP header to the request.

When you update an item, the response is in the same format as when you read an item. You can also use the `fields` and `payload` HTTP parameters to customize the response.

Important
Due to a known issue, that is related to the WebDAV functionality in Sitecore, and that exists in Sitecore CMS versions prior to 6.5.0 Update 5, you must disable the `WebDAV.Enabled` setting in the `Sitecore.WebDAV.config` file to be able to use the `UPDATE` and `DELETE` operations.

Updating Existing Items

The HTTP `PUT` method is used to update existing items. This affects all items in the resolved scope. To update the item fields in the scope, you must specify the field data in the request body in the standard format.

The field values can contain special characters and should therefore be encoded when they are passed to the Item Web API.

You must specify the fields in the PUT request body in one of the following formats:

- `<fieldName1>=<fieldValue1>&<fieldName2>=<fieldValue2>&<fieldNameN>=<fieldValueN>`
- `<fieldID1>=<fieldValue1>&<fieldID2>=<fieldValue2>&<fieldIDN>=<fieldValueN>`

If an item contains a field with the specified name, the field is updated with the value that you specify.

Note
The field names are not case sensitive and the *update* operation returns a collection of the updated items.

When you want to update a field, you must add the `Content-Type=application/x-www-form-urlencoded` HTTP header to the request.

Deleting Existing Items

Use the HTTP `DELETE` method to delete existing items. It affects all the items in the resolved scope. The response contains the number of deleted items and their IDs. This response does not include any descendent items that were deleted — it does not count them or list their IDs.

Creating Media Items

Use the HTTP `POST` method to upload media files to the Media Library. You can create several media items at the same time.

This operation does not use the items in the scope. However, it uses the context item as a parent for the media items that are being created. The context item is resolved by its path or ID and should be the Media Library root or one of its descendants.

The only parameter is the name of the media items to be created — `name` HTTP parameter. If you upload several files at once, unique names based on the specified name of each item are applied to the media

items. You do not need to specify the template because the Media Manager uses the content type of the uploaded file to resolve it.

Right now, the Item Web API can only create unversioned media items. You cannot use the Item Web API to create a versioned media item.

The response contains the created media items in the same format as they are returned by the *read* item operation.

When you upload a media item, you must add the `Content-Type=multipart/form-data` HTTP header to the request.

Security

The Sitecore Item Web API supports:

- The standard Sitecore security system.
- Advanced Item Web API security.

By default, the Item Web API is disabled for all managed sites. Before you enable it, you must decide which security model is appropriate for each particular managed site in your Sitecore solution.

To set the security model for a managed site, open the `Sitecore.ItemWebApi.config` file and modify the following section:

```
<site name="<SiteName>">
  <patch:attribute name="itemwebapi.mode"><Mode></patch:attribute>
  <patch:attribute name="itemwebapi.access"><Access></patch:attribute>
  <patch:attribute name="itemwebapi.allowanonymousaccess"><AllowAnonymousAccess>
  </patch:attribute>
</site>
```

where:

- The `<Mode>` attribute can take the following values:
  - `Off` — the Item Web API is turned off.
  - `StandardSecurity` — the Item Web API is turned on and the default Sitecore security model is used.
  - `AdvancedSecurity` — the Item Web API is turned on, and the default Sitecore security model is extended with the requirement to set the `remote:fieldread` access right for content fields.
- The `<Access>` attribute can take the following values:
  - `ReadOnly` — to only allow the READ operation.
  - `ReadWrite` — to allow the CREATE, READ, UPDATE, and DELETE (CRUD) operations.
- The `<AllowAnonymousAccess>` attribute can be set to `true` or `false`. It specifies if is for non-authenticated users have access.

Examples:

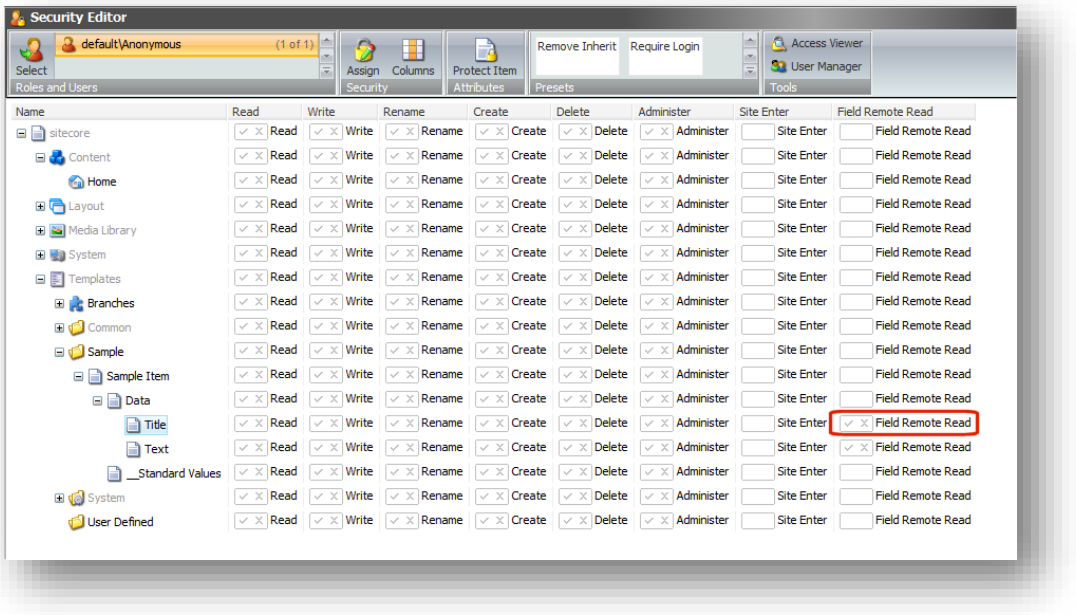
```
<site name="shell">
  <patch:attribute name="itemwebapi.mode">StandardSecurity</patch:attribute>
  <patch:attribute name="itemwebapi.access">ReadWrite</patch:attribute>
  <patch:attribute name="itemwebapi.allowanonymousaccess">false</patch:attribute>
</site>

<site name="website">
  <patch:attribute name="itemwebapi.mode">AdvancedSecurity</patch:attribute>
  <patch:attribute name="itemwebapi.access">ReadOnly</patch:attribute>
  <patch:attribute name="itemwebapi.allowanonymousaccess">true</patch:attribute>
</site>
```

For content delivery servers, we recommend the following configuration settings:

- `<Mode>=AdvancedSecurity`
- `<Access>=ReadOnly`
- `<AllowAnonymousAccess>=false`

The default Sitecore security model has been extended to support the Item Web API requests:

1. The *Field Remote Read* field-level access right was introduced.



   In contrast with the *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`.

2. If the Item Web API request targets a site within the domain of Sitecore security, for example shell site, the client should be logged in the same domain. Otherwise, the Item Web API responds with a `Request forbidden 403`.

   The Item Web API also responds with `Request forbidden 403` if:

   - An anonymous client makes a request to a site that requires a login.
   - The client making the request has not been assigned the *Site Enter* access right.

User Authentication

The user credentials should be passed in the headers of the HTTP request as follows:

- `X-Scitemwebapi-Username` — the user name
- `X-Scitemwebapi-Password` — the user password

The authentication is stateless, which means that the user credentials are passed with each request.

In the Item Web API, you can use cookies for authentication. These authentication cookies are useful when using the API in the Sitecore backend.

This only works when the user has already been given the authentication cookies. However, the Item Web API does not contain a method that allows users to log in and receive the authentication cookies.

Note
When you use the authentication cookies, you must not pass the credentials in the HTTP headers.

Credentials Encryption

The Sitecore Item Web API supports the encryption of the credentials.

To encrypt the credentials, you should:

1. Use the `getpublickey action` to get the Item Web API encryption public key:

   `http://<host_name>/-/item/v1/-/actions/getpublickey`.

   The key is then returned as plain text in XML DSig format.

2. Use the RSA algorithm to encrypt both the user name and the password with a public key. These encrypted credentials should be in UTF-8 format and Base64 encoded.

3. Set the `X-Scitemwebapi-Encrypted` HTTP header to `1` in the request.

If you use an SSL connection, the credentials must not be encrypted. The server takes care of header encryption. We recommend that you use an SSL connection instead of custom encryption.

By default, the Item Web API server uses a 1024-bit encryption key. In the `Sitecore.ItemWebApi.config` file, you can also use the `keyLength` setting to specify the value of the RSA encryption provider.

The private encryption key is stored in the machine key storage. The name of the default key container is `SCWEBAPIKEYCONTAINER`. You can use the `keyContainer` property of the RSA encryption provider to change the name of the default key container.

The Sitecore Item Web API uses PKCS#1 v1.5 padding for encryption and decryption.

For more information, see `http://msdn.microsoft.com/en-us/library/system.security.cryptography.rsacryptoserviceprovider.encrypt.aspx`

Special Requests

The Item Web API also handles special requests that are not related to CRUD operations. These requests have response formats that are different from the regular Item Web API requests. The response format and the content type vary according to the operation.

A special request must have the following format:

```
http://<host_name>/<webapi_hook>/<special_request_hook>/<action_name>[?<optional_parameters>]
```

Where:

- `<special_request_hook>` can be assigned to `-/actions`
- `<optional_parameters>` can be assigned to `[<key1>=<value1>][&<key2>=<value2>]...[&<keyN>=<valueN>]`
- `<optional_parameters>` is a regular HTTP query string.
- `<action_name>` is the name of the operation. The action name is not case sensitive.

Example:

```
http://<host_name>/-/item/v1/-/actions/GetRenderingHtml?database=master&language=en&renderingId={493B3A83-0FA7-4484-8FC9-4680991CF743}&itemId={110D559F-DEA5-42EA-9C1C-8A5DF7E70EF9}&itemVersion=7&a=1&b=2&c=3
```

`GetRenderingHtml` is the action name. The content type of the response to a special request varies according to the operation type.

Getting Rendered HTML

The `GetRenderingHtml` request gets the output HTML of a rendering.

The required parameters are:

- `sc_database` — the database that contains the rendering and source items.
- `renderingId` — the ID of the rendering item.
- `sc_itemId` — the ID of the source item.

The optional parameters are:

- `sc_itemversion` — the version of the source item. If not specified, the `Version.Latest` value is used.
- `language` — the language of the rendering and the source items. If not specified, the default language is used.

All the other query string parameters are recognized as the rendering parameters, such as `a=1&b=2&c=3` in the following example:

```
http://<host_name>/-/item/v1/-
/actions/GetRenderingHtml?sc_database=master&language=en&renderingId={493B3A8
3-0FA7-4484-8FC9-4680991CF743}&sc_itemid={110D559F-DEA5-42EA-9C1C-
8A5DF7E70EF9}&a=1&b=2&c=3
```

Note
The `database`, `itemId`, `itemversion`, and `sc_lang` parameter names are reserved keys in this version of the Item Web API and you cannot use them as rendering parameters.

Extending the Sitecore Item Web API

Most of the Sitecore Item Web API features are implemented using pipelines. You can remove existing pipeline processors, override default processors, and add your own processors to customize the Item Web API, without changing the default functionality.

The Item Web API uses the following pipelines:

itemWebApiRequest

This is the main pipeline that represents the lifecycle of an Item Web API request. You can use this pipeline for many purposes, such as modifying the item set for which the CRUD operation is going to be executed, and producing the output in a different format.

The following snippets illustrate the class and the processor that are used to produce the output in XML format.

The `XMLSerializer` class:

```
internal class XmlSerializer : ISerializer
{
  public string SerializedDataMediaType
  {
    get
    {
      return "text/xml";
    }
  }

  public string Serialize(object value)
  {
    return Serialize((Dynamic)value).ToString();
  }

  private XElement Serialize(Dynamic value)
  {
    var element = XElement.Parse("<object/>");

    foreach (var property in value)
    {
      var propertyName = string.Format("_{0}", property.Key);
      propertyName = propertyName.Replace("{", "");
      propertyName = propertyName.Replace("}", "");
      propertyName = propertyName.Replace(" ", "-");

      var child = XElement.Parse(string.Format("<{0}/>", propertyName));
      var array = property.Value as Dynamic[];

      if (array != null)
      {
        foreach (var item in array)
        {
          child.Add(Serialize(item));
        }
      }
      else if (property.Value is Dynamic)
      {
```

```
            child.Add(Serialize((Dynamic) property.Value));
          }
          else
          {
            child.Add(property.Value.ToString());
          }

          element.Add(child);
        }

        return element;
      }
    }
```

The `SwitchToXmlSerializer` processor:

```
    public class SwitchToXmlSerializer : RequestProcessor
    {
      public override void Process([NotNull] RequestArgs arguments)
      {
        Context.Current.Serializer = new XmlSerializer();
      }
    }
```

The `SwitchToXmlSerializer` processor must precede the default `SerializeResponse` processor in the `Sitecore.ItemWebApi.config` file.

itemWebApiCreate

This pipeline runs the *create* item operation.

When the item is created, it invokes the `itemWebApiRead` pipeline to read the item's fields and properties.

itemWebApiRead

This pipeline runs as part of the *read* item operation.

To get the item fields and properties, this pipeline invokes the `itemWebApiGetFields` and the `itemWebApiGetProperties` pipelines respectively.

itemWebApiUpdate

This pipeline runs as part of the *update* item operation.

When item is updated, it invokes the `itemWebApiRead` pipeline to read the updated item's fields and properties.

itemWebApiDelete

This pipeline runs as part of the *delete* item operation.

itemWebApiGetFields

This pipeline retrieves the fields in an item.

The following snippet illustrates the `RemoveSharedFields` processor that is used to remove all the shared fields from the response:

```
    public class RemoveSharedFields : GetFieldsProcessor
    {
      public override void Process(GetFieldsArgs arguments)
      {
        foreach (var field in arguments.Fields)
        {
          if (field.Shared)
          {
```

```
            arguments.RemoveField(field);
          }
        }
      }
    }
```

The `itemWebApiGetFields` pipeline should come after the default `GetFields` processor in the `Sitecore.ItemWebApi.config` file.

itemWebApiGetProperties

This pipeline retrieves the item's properties.

The following snippet illustrates the `CustomizeProperties` processor that is used to extend the Item Web API response with custom item properties, such as `IsClone`:

```
public class CustomizeProperties : GetPropertiesProcessor
{
  public override void Process(GetPropertiesArgs arguments)
  {
    var item = arguments.Item;
    var properties = arguments.Properties;
    properties.Add("IsClone", item.IsClone);
  }
}
```

The `CustomizeProperties` processor should come after the default `GetProperties` processor in the `Sitecore.ItemWebApi.config` file.