# Requesting UDC data

**URL:** https://docs.grassfish.com/docs/requesting-udc-data

**Archiviert am:** 2025-07-17 18:36:30

Requesting dynamic data via the UDC interface is simple with the gfUdcConnector. Use the following instructions to integrate the connector in HTML Basic and HTML Wizard spots.

## Including the library

The gfUdcConnector is embedded in the index.html page via the „script“ tag, e.g.: **<script src="PATH_TO_SCRIPT/gfUdcConnector.js"></script>**

## Instantiating the gfUdcConnector

In order to allow the invocation of the gfUdcConnector function, it must be instantiated within the particular code prior to this:

```
 var udc = new GFUdcConnector();

udc.getVersion();
```

## Object typification

### UDC data objects

In order to better illustrate the structure of the data, individual objects are typified and used for descriptions of functions in the following sections.

#### UdcData

```
 UdcData: {
   Success: boolean,
   Result: DataSource[]
};
```

| Property | Description |
|---|---|

| | |
|---|---|
| Success | true: valid data is available<br><br>false: error encountered during data request |
| Result | Array of DataSource objects which relate to data sources linked to the spot (see Adding a data source). Therefore, the number can also be "0..n". |

**UdcDataByKey**

```
 UdcDataByKey: {
   Success: boolean,
   Result: DataSource
};
```

| **Property** | **Description** |
|---|---|
| Success | true: valid data available<br><br>false: error encountered during data request |
| Result | The DataSource object which was resolved via the description.<br>undefined, if no data source was found for the corresponding description. |

**DataSource**

```
 DataSource: {
   Categories: Category[],
   ContainerGuid: string,
   Elements: Element[],
   Key: string,
   LastModified: string,
   PluginInstanceName: string
};
```

| **Property** | **Description** |
|---|---|
| Categories | Array of 0..n Category objects |
| ContainerGuid | Distinct identification of the data source (specific to the spot or spot instance) |
| Elements | Array of 0..n Element objects |

| | |
|---|---|
| Key | Description of the data source. This value relates to the one assigned to the description of the data source in CMS (specific to the spot or spot instance). |
| LastModified | Last modification |
| PluginInstanceName | Name of the data source |

**Category**

```
  Category: {
    ExternalId: string,
    Id: number,
    LastModified: string,
    MediaFiles: any[]
    Name: string
 };
```

| Property | Description |
|---|---|
| ExternalId | Distinct identification |
| Id | Internal key for logical relationships with the Element objects. |
| LastModified | Last modification |
| MediaFiles | Array of file information. These depend on the applicable plugin and are not described further here. |
| Name | Name of the category |

**Element**

```
  Element: {
    CategoryIds: number[],
    ElementValues: any[],
    ExternalId: string,
    Guid: string,
    Id: number,
    LastModified: string,
    MediaFileBaseUrl: string,
    MediaFileDirectory: string,
    MediaFiles: any[],
    Name: string
};
```

| Property | Description |
|---|---|
| CategoryIds | Combination with 0..n Category objects |
| ElementValues | Object with the applicable information in form of a list of key-value pairs. These are already typified but depend on the corresponding plugin and are not described here further. |
| ExternalId | Distinct identification |
| Id | Internal key |
| LastModified | Last modification |
| MediaFileBaseUrl | URL to UDC media on the server/player |
| MediaFileDirectory | Directory of the file system on the server/player |
| MediaFiles | Array of 0..n MediaFile objects |
| Name | Name of the element |

**MediaFile**

```
  MediaFile: {
    Image: string
};
```

| Property | Description |
|---|---|
| Image | File name of the medium |

## Debug objects

The determination of the parameters required for the test connection is described in detail in Setting up a UDC test connection.

#### DebugServer

```
DebugServer = {
  url: string,
  locationId: number,
  spotId: number,
  sessionId: string
};
```

| Property | Description | Example |
|---|---|---|
| locationId | Internal key of the player | 65 |
| sessionId | Valid session key for the authentication | d3872316-dc92-48e8-8659-341084b2ee12 |
| spotId | Internal key of the spot | 697 |
| url | Address to server web services | http://{DOMAIN}/GV2/Webservices/rest/gui/api/ |

#### DebugClient

```
DebugServer = {
  ip: string,
  siid: number
};
```

| Property | Description | Example |
|---|---|---|
| ip | IP address of the player which should be tested | 192.168.169.73 |
| siid | Internal key of the spot instance | 529 |

**DebugFile**

```
 DebugServer = {
   url: string
};
```

| Property | Description | Example |
|---|---|---|
| url | Relative path to locally available UDC data | ./mock/udc.json |

## Available features

### getData

By means of this function it is possible to retrieve all UDC data connected to the spot.

**Parameters:**

```
 {callback} onSuccess: returns UdcData, result as plain text {string}
{callback} onError: returns error message {string}
```

**Example**:

```
 var udc = new GFUdcConnector();
udc.getData(function(jsonData, plainData)
{
   // data as JSON {UdcData}
   console.log(jsonData);
   // data as plain text {string}
   console.log(plainData);

}, function(message)
{
   console.error(message);
});
```

**Explanation:**

The getData function is assigned with two call-back functions. If successful, the first function is invoked with the UdcData object and data as a string (see UdcData). In case of an error, the second function is invoked with the error message.

UdcData.Result is an array of DataSource objects which relate to the data sources which are linked to the spot or spot instance. Depending on how many data sources are assigned to the spot or spot instance, the number amounts to 0..n objects.

The individual data sources then must be identified via the description of the data source (key) and processed accordingly.

Note

If only few data sources are linked to the spot, it is recommended to retrieve data directly via the GetDataByKey function in order to avoid the subsequent identification.

However, at the same time this increases the number of HTTP requests and thus the system load.

## getDataByKey

By means of this function it is possible to request a specific data source via its unique description.

Note

For a multitude of linked data sources, it is recommended to retrieve the data sources once with getData and then process it. This conserves the system load.

**Parameters:**

```
 {string} key: unique designatorht

{callback} onSuccess: returns UdcDataByKey, result as plain text {string}
{callback} onError: returns error message {string}
```

**Example**:

```
 var udc = new GFUdcConnector();
udc.getDataByKey("News", function(jsonData, plainData)
{
   // data as JSON {UdcData}
   console.log(jsonData);

   // data as plain text {string}
   console.log(plainData);

}, function(message)
{
   console.error(message);
});
```

**Explanation**:

In addition to the unique description of the data source, the getDataByKey function is assigned with two call-back functions. If successful, the first function is invoked as a string with the UdcData object and data (see UdcData). In case of an error, the second function is requested with the error message.

UdcDataByKey.Result is a DataSource object which relates to a data source assigned to a spot or spot instance. If a data source cannot be found via its unique description, it is UdcDataByKey.Result undefined.

## setLocalTestData

By means of this function it is possible to test a spot against various UDC web services and goals during development. The parameters of the individual connection types must be set correctly in order to ensure the error-free operation of the interface.

Note

This functionality is primarily intended for development and must be commented out or removed entirely when going live. If not, it is possible to encounter unwanted erroneous behaviour.

**Parameters:**

*{DebugServer|DebugClient|DebugFile} connection*

**Example:**

```
 var udc = new GFUdcConnector();
var connection = {
   ip: "192.168.169.73",
   siid: 529
};
udc.setLocalTestData(connection);
// fetch some data...
```

**Explanation:**

Depending on the transferred connection type, data is requested from the corresponding source and then made available. For this, the following distinction is made:

- Testing against server web services, handover of DebugServer object
- Testing against player web services, handover of DebugClient object
- Testing against a local file, handover of DebugFile object

Further information can be found under Setting up a UDC test connection.

## registerLogHandler

By means of this function, it is possible to request additional logs via the gfUdcConnector.

**Parameters:**

*{callback} onLog: returns log messsag {string}*

**Explanation:**

```
 var udc = new GFUdcConnector();
udc.registerLogHandler(function(message)
{
   console.log(message);
});
```

**Explanation:**

The registerLogHandler function is assigned with a call-back function as a parameter which in case of an entry, is invoked via the gfUdcConnector. The log notification is rendered as a parameter.