# AngularJS webforms - getting started

Updated 2014-10-20

Building webforms using [AngularJS](https://angularjs.org/) allows us to build highly flexible, performant and interactive forms. One of the benefits with AngularJS from the developers’ perspective is the data binding, which makes it possible to work on an in-memory representation of an instance of the XML Schema created for the reporting in an HTML form, that when finished editing can be saved back into its XML representation again.

## Overview

The related tasks when implementing a new webform for reporting begins with creating a XML schema, matching the reporting requirements. The schema is to be uploaded and published on the [Eionet data dictionary](http://dd.eionet.europa.eu/) (DD). Together with the schema, an empty instance file of the schema needs to be generated.

The next step is to produce the webform, which on a basic level should load this instance of the schema from the server, let the user edit the elements/fields using a regular HTML form, and later save it back to the server. Since the AngularJS webforms interacts better with JSON instead of XML, the instance file is converted on the server to JSON when loaded, and back again when saving.

When the webform is produced, it should be uploaded to the [Eionet web questionnaires platform](http://webforms.eionet.europa.eu/) (WebQ). In order to connect the webform with the normal reporting procedures by countries, via the envelopes on [Eionet Central Data Repository](http://cdr.eionet.europa.eu/) (CDR), the schema published on DD earlier has to be mapped to the correct reporting obligation in the [Reporting Obligations Database (ROD)](http://rod.eionet.europa.eu/) and also to the webform. The schema–reporting obligation mapping is done in CDR, and the mapping between the schema and the webform is done in WebQ.

Creating the mapping makes CDR recognise that there is a webform in place for this obligation and therefore provide a link for the user to edit an instance file using the webform. On the first edit the instance file will be created and added to the envelope from WebQ, and subsequent edits will update this file. Basically the webform plugs into CDR as a means for editing the XML-file in the envelope, leaving the other steps in the CDR reporting workflow as normal.

## Creating a webform

The quickest route to understand how to create a new webform is probably to look at some of the current ones in the [Dataflows subversion repository](https://svn.eionet.europa.eu/repositories/Reportnet/Dataflows/). For example the Industrial Emissions Directive (IED) [webform](https://svn.eionet.europa.eu/repositories/Reportnet/Dataflows/IndustrialEmissionsDirective/) is quite simple, while the MMR Article 17 [webform](https://svn.eionet.europa.eu/repositories/Reportnet/Dataflows/MMR-Article17/webform/) provides an example of a more advanced tabular form. In principle, a suitable html-form covering the required information specified by the XML schema has to be developed, and the fields has to be hooked into the AngularJS-data model. The next step is to integrate it with the WebQ platform.

Apart from the setup procedure explained further down, the specifics of integrating the webform with WebQ are mainly about how to load and save data from and to WebQ. When someone opens a webform it will be called with a set of parameters:

* The values for “base\_uri” and “envelope” should be used to get the path for loading additional files, and also return the user to the envelope after closing the webform.
* The parameter “fileId” is the identifier for the instance file (new or existing one), and a GET-request needs to be made back to WebQ using the URL “/download/converted\_user\_file?fileId=*<fileId parameter value>*”, which will return the user instance file formatted (converted) into JSON. If the user is just starting the reporting in a new envelope without an instance file created yet, WebQ will return a new copy of the empty instance file (created together with the schema earlier).
* Finally, “sessionid” should be appended to the URL on calls for loading and saving the instance data.

After receiving the JSON representation of the instance file, it can now be used as the data model in AngularJS, binding it to the input fields in the HTML-form. When the user has finished editing the form (or just wants to save it), the updated data model can be saved back to the server (envelope) by issuing a HTTP POST-request with the JSON-representation of the instance file to the URL “/saveXml?fileId=*<fileId parameter value>*”. When the user closes the webform he/she should be redirected back to the envelope-URL.

Looking at some of the previous EEA webforms developed with AngularJS, the other main parts of the code are mainly about implementing eventual translations of question labels and code lists, which usually are kept in separate files, informing the user if actions takes time to complete (such as loading the webform), preventing the user from accidentally leaving the form with unsaved data, and of course validation.

## XML – JSON conversion

For understanding how the XML-file is represented in the JSON-format received on the client side from the conversion on WebQ, investigating the library used in WebQ ([Staxon](https://github.com/beckchr/staxon/)), and [the mapping convention](https://github.com/beckchr/staxon/wiki/Mapping-Convention) could be useful. To avoid the common pitfalls, there are some things to keep in mind when doing the XML-JSON conversion:

* If the form has fields, or rows of fields that can be added or removed, measures have to be taken before saving it back to the server to make sure the order of the fields still match the original XML instance file as JavaScript does not guarantee the order of object properties.
* All the attribute properties (starting with “@”) have to go after the main element. Especially the root element attributes cannot be added to the end of the JSON model.
* The automatic XML to JSON conversion does not create array object if only one element exists. They have to be converted to arrays in the webform JavaScript code.

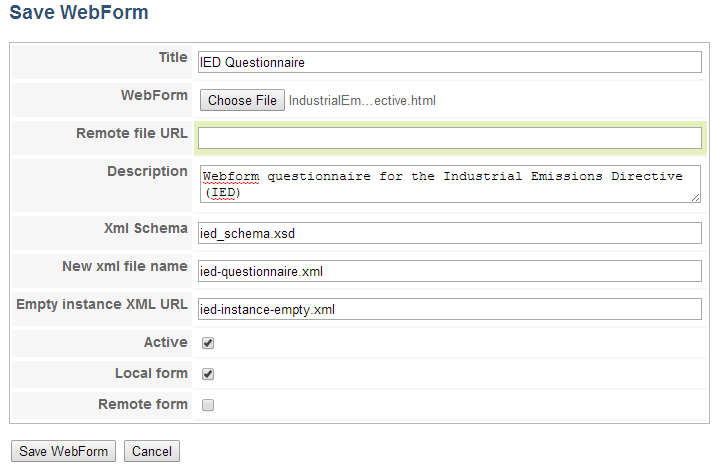
The WebQ service also provides additional methods that can be useful when producing a webform:

* XML-files in the webform project folder can be downloaded in JSON format by appending a “format=json”-parameter to project file URL, e.g. “http://webforms.eionet.europa.eu/download/project/ied/file/ied-instance-empty.xml?**format=json**”
* Additional information about a webform can be retrieved by calling the URL “/file/info?fileId=<*fileId parameter value*>”, e.g. “http://webq2test.eionet.europa.eu/file/info?fileId=*2674*”. This information will be in XML format though.
* Any conversion service registered with the [Conversion Service](http://converters.eionet.europa.eu/) via the webforms underlying XML Schema can be used by calling the url: “/download/convert?fileId=<*fileId parameter value*>&conversionId<*conversionId>*”.The list of conversions registered for the webform can be found in the response from the Info-call described above.

## Using code lists

The preferred way to implement the code lists often needed for the webform’s question labels and values for fixed choices, is to have these lists published on DD in JSON-LD format and use them directly from there in the webform. One example that is simple and understandable is the [countries vocabulary](http://dd.eionet.europa.eu/vocabulary/common/countries/json), where the consuming webform needs to iterate over the list of concepts to retrieve each country-ID from the “@id”-property and the language specific label from the “@prefLabel”-property. It is possible to limit the result by appending a “lang”, “id” or “label” parameter to the URL. For how this can be consumed through a webform, see the [Habides webform](https://svn.eionet.europa.eu/repositories/Reportnet/Dataflows/Habides/angular/).

## Setting up a webform for test

* An Eionet account is needed, if you don’t have one already, send an email to [helpdesk@eionet.europa.eu](mailto:helpdesk@eionet.europa.eu) and cc someone who can authorise this request.
* Log on with your Eionet account on the web questionnaires test platform at [webq2test.eionet.europa.eu](http://webq2test.eionet.europa.eu/).
* Select “Webform projects” from the menu and add your project from the “Operations”-menu.
* Enter a unique name without spaces as project ID, a descriptive name, and save it.
* The project is added to the list, click on it to open it. From the “Operations”-menu select add file and upload the empty instance file, the schema and finally any javascript-, css- and other files referenced from the webform. They will be accessible from the same path as the webform.
* From the “Operations”-menu, now select “Add webform” and fill out the form - uploading your HTML-webform and referencing the schema and empty instance file added before. Remember to tick “Active” and “Local form” so it can be used for testing.  
  
* Try out your newly added webform by selecting “My session files” from the menu to the left. Select “Create new session file”, choose the one added from the list, and select “Save file in session”. A new file will be added to your list of session files, which can be opened in your webform by selecting “Edit with '<your webform name>' web form” in the “Actions”-column.

## Setting up a webform for production

This is similar to setting up the webform for test, but requires additional steps to integrate with the other parts of CDR.

* Instead of uploading the schema to the project on WebQ, the schema needs to be added in the [Eionet data dictionary](http://dd.eionet.europa.eu/) (DD).
* The mapping between the reporting obligation in ROD and the schema now published in DD needs to be setup in CDR.

Contact relevant EEA staff for any of these steps if you don’t have access.