Deliverable Number: D8.1

Deliverable Title: **API for databases + explanatory note**

Work Package: WP8

Deliverable type: Other

Dissemination status: Public

Submitted by: SHARE ERIC

Authors:

Maurice Martens (SHARE, CentERdata)

Date Submitted: January 2018

This project has received funding from the *European Union's Horizon 2020 research and innovation programme* under grant agreement No 654221.

www.seriss.eu @SERISS_EU

SERISS (Synergies for Europe's Research Infrastructures in the Social Sciences) aims to exploit synergies, foster collaboration and develop shared standards between Europe's social science infrastructures in order to better equip these infrastructures to play a major role in addressing Europe's grand societal challenges and ensure that European policymaking is built on a solid base of the highest-quality socio-economic evidence.

The four year project (2015-19) is a collaboration between the three leading European Research Infrastructures in the social sciences – the European Social Survey (ESS ERIC), the Survey of Health Ageing and Retirement in Europe (SHARE ERIC) and the Consortium of European Social Science Data Archives (CESSDA AS) – and organisations representing the Generations and Gender Programme (GGP), European Values Study (EVS) and the WageIndicator Survey.

Work focuses on three key areas: Addressing key challenges for cross-national data collection, breaking down barriers between social science infrastructures and embracing the future of the social sciences.

Please cite this deliverable as: Martens, M. (2017) API for databases + explanatory note. Deliverable 8.1 of the SERISS project funded under the *European Union's Horizon 2020 research and innovation programme* GA No: 654221. Available at: www.seriss.eu/resources/deliverables

Table of content

Executive summary **3**
1. Introducing SERISS **4**
2. Functionality of the API in web surveys (CAWI mode) **4**
2.1.1 Common arguments 5
2.1.2 Mode 1: List of countries 5
2.1.3 Mode 2: Search (used in autocomplete) 5
2.1.4 Mode 3: Tree view 6
2.1.4: Mode 4: Retrieving data of specific coding 6
3. How can survey holders use the API in CAWI? **7**
4. How can survey holders use the API in CAPI? **24**
5. References **26**

Executive summary

WP8's coding module for socio-economic survey questions' seeks to harness technological improvements and provide tools that can greatly increase the efficiency and reliability of the coding of open text responses in social surveys. Occupation, industry, employment status, educational attainment, field of education, and social inclusion are five core variables in most socio-economic and health surveys. However, their measurement, especially in a cross-cultural, cross-national and longitudinal context, is cumbersome, not sufficiently standardized and typically asked open-ended, followed by semi-automatic office coding, employing dictionary approaches (coding indexes), with subsequent manual coding of the cases that could not be coded automatically. Office coding is time-consuming and costly, while coding comparability across countries (or even across coders within countries) is not well-established.

Recent innovations have put forward alternatives to office coding. The use of web surveys is fast increasing and these surveys allow respondents' self-identification, using search trees with (nested) long lists of responses or using semantic matching technologies employing (very) large dictionaries. Such technologies can also be employed in face-to-face or telephone surveys, because these surveys are increasingly administered using tablets or laptop computers. Semantic technologies can increasingly be supplemented with machine learning algorithms and with statistical matching. These are all promising improvements towards high quality identification and subsequent classification of the core variables.

SERISS has funded the development of APIs (Application Programming Interface) to allow for self-coding of socio-economic variables by survey respondents. Three variables – occupation, industry and educational attainment - are currently available as an API from the website http://surveycodings.org/. This deliverable briefly describes how the APIs can be used in web surveys and what is needed to use them for computer-based face-to-face or telephone interviews.

1. Introducing SERISS

Synergies for Europe's Research Infrastructures in the Social Sciences (SERISS) is a four-year project that aims to strengthen and harmonise social science research across Europe (2015-19). Work Package 8 (WP8) of SERISS aims to provide cross-country harmonised, fast, high-quality and cost-effective coding of open ended questions on respondents' occupations, industries and education into international standardized classification systems, and to develop a tool to collect standardized social network information. Occupation, industry, employment status, educational attainment and field of education are core variables in many socio- economic and health surveys. Moreover, the size and intensity of social networks are key variables in social surveys. However, their measurement, especially in a cross-cultural, cross-national and longitudinal context, is cumbersome, not sufficiently standardized and often expensive. This work package takes recent scientific and technological developments as an opportunity to improve this situation in order to improve survey measurement quality and provide cost-effective solutions to Research Infrastructures (SERISS Annex 1, European Commission, 2015).

Building on the current technology and the partners' experiences, WP8 develops a cross-country harmonised, fast, high-quality and cost-effective coding module for the core variables. The module and its APIs use a large multi-lingual dictionary with tens of thousands of entries about job titles, industry names, fields of education and training, and employment status categories. Additionally, the module includes country-specific, structured lists of educational qualifications. The module provides up-to-date codes to classify the variables, using international standardized classification systems. It facilitates surveys in the ESS, EVS, GGP, SHARE and WageIndicator countries and their associated networks to serve infrastructures reaching out to a global audience, including the most spoken languages outside the EU28 area, notably Russian, Mandarin, Arabic, Hindi and Bahasa. WP 8 covers in total 47 languages servicing 99 countries. For the choice of countries and languages, see Deliverable D8.14 (Tijdens, 2016b).

This report concerns deliverable D8.1 of WP8, part of Task 8.1: "Programming the module" (SERISS Annex 1, European Commission, 2015). The responsible partner is SHARE CentERdata. Task 8.1 consists of two deliverables. D8.1 is a software deliverable concerning the API, and D8.2 is the software needed to run the module with the survey questions related to the five core variables.

2. Functionality of the API in web surveys (CAWI mode)

Questionnaires that have an internet connection can call the surveycodings.org functionality by making calls to the server at https://api.surveycodings.org. By attaching arguments to this address, commands can be given to this server. We can currently call three types of coded sets: educational attainment, occupations and industries. For each set four modes are available: country, search, tree, coding.

There are four modes of data retrieval. Each mode is used in different situations. You can view a complete example when inspecting the html and JavaScript code on the Database live search pages throughout the website.

- Each mode has a specific set of combination of options. Please see the description of each mode below for the correct options.
- **Always** tell the API what database to use. This is done with the **type** argument. See below for details.

- The API returns all data in JSON format.
  The first argument starts with a question mark: ***?***, the following arguments start with ***&***. See the examples on how to write a complete URL correctly.

2.1.1 Common arguments

Argument	Options	Description	Example
Type	education occupation industry	Tells the API in which database to search the codings. There are **three** different databases with codings. With argument **type=[database]** you can tell the API which database to use. **Always** use this argument.	type=education type=occupation type=industry
Mode	country search tree coding	Tells the API what to do.	mode=country mode=search mode=tree mode=coding

2.1.2 Mode 1: List of countries

Tell the API to return a JSON format of all **countries** in the **education, industry or occupation** databases that have codings. The arguments **type** and **mode** are used to retrieve the requested data. See the above section on what they do.

Example: https://api.surveycodings.org/codings/search.json?type=education&mode=country

Argument	Options	Description	Example
Mode	country	The above URL tells the API to return a JSON format of all **countries.**	mode=country

2.1.3 Mode 2: Search (used in autocomplete)

Tell the API to return a JSON list of all codings for a specific context where the description matches a given search string.

Example education database:
https://api.surveycodings.org/codings/search.json?type=education&mode=search&context=52800&search=pr

Example occupation/industry database:
https://api.surveycodings.org/codings/search.json?type=occupation&mode=search&context=en_GB&search=pr
https://api.surveycodings.org/codings/search.json?type=industry&mode=search&context=en_GB&search=pr

Argument	Options	Description	Example
Mode	search	Tells the API to search for codings based on a text search.	mode=search

Context	[integer] or [string]	This tells the database in which context to look for codings. The context for education is an integer. The context for industry and occupation are strings.	Example education: **context=52800** Dutch codings. Example occupation: **context=nl_NL** Dutch codings. Example industry: **context=nl_NL** Dutch codings.
Search	[string]	A string that will be broken down into parts. Any (partially) match on any parts will return results. Multiple parts are allowed. For example 'search=a b' will return all results with either an 'a' or 'b' in it.	search=tea search=any string

2.1.4 Mode 3: Tree view

Tell the API to return a JSON tree representation of all codings for a specific context.

Example structure on how to call the education database:
https://api.surveycodings.org/codings/search.json?type=education&mode=tree&context=52800

Example occupation/industry database:
https://api.surveycodings.org/codings/search.json?type=occupation&mode=tree&context=nl_NL
https://api.surveycodings.org/codings/search.json?type=industry&mode=tree&context=nl_NL

Argument	**Options**	**Description**	**Example**
Mode	tree	Tells the API to return a full tree of all codings.	mode=tree
Context	[integer] or [string]	This tells the database in which context to look for codings. The context for education is an integer. The context for industry and occupation are strings.	Example education: **context=52800** Dutch codings. Example occupation: **context=nl_NL** Dutch codings. Example industry: **context=nl_NL** Dutch codings.

2.1.4: Mode 4: Retrieving data of specific coding

Tell the API to return detailed data about the requested coding.

https://api.surveycodings.org/codings/search.json?type=education&mode=coding&coding=52800079

Returns an empty set if the coding identifier is unknown.

Argument	Options	Description	Example
Mode	coding	Tells the API to return the details of a specific coding.	mode=coding
Coding	[integer]	Tells the database which coding is requested. This is the name of the coding.	coding=528000

3. How can survey holders use the API in CAWI?

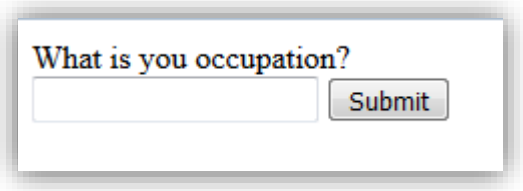
In socio economic questionnaires, often occupational title, industry or education are asked for. This might be fun for a respondent to talk about, but without a code from some classification its use is very limited. Once classified, researchers can do statistics. This section will focus on the technique behind web questionnaires. It will start at a very basic level to explain the survey process from technical perspective and will explain how to integrate the surveycodings into this.

There are several techniques to integrate the surveycodings lists in your own web questionnaire. Call the surveycodings service, make your own service, or send the complete lists over to the respondent's computer and do not use a service at all.

Web questionnaires are questionnaires that display in a browser. It uses html code to show the questions and has a form that contains the response options of that question. In the remainder of this document, we will stick with coding of occupation as an example. Industry coding and education coding, work similar, if there are any deviations they will be pointed out.

Example html code for a question could be:

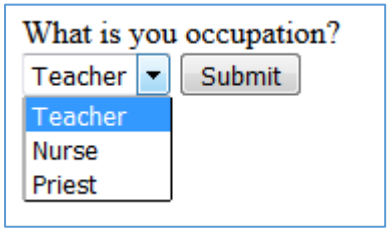
```
<html>
	<body>
		What is you occupation?
		<form>
			<input name="occupation" type="text" />
			<input type="submit" value="Submit" />
		</form>
	</body>
</html>
```



Normally code like this will be generated by the survey system, hidden in a lot more code to make it look and feel astounding, but the basis for web questionnaires comes down to this. If a respondent answers this question, we will get an open text answer. To assign a classification to this answer, an expert coder or a system should compare this answer with a list of occupations and determine what the most likely classification is.

An alternative approach to determining the occupation is to use a list.

```
<html>
	<body>
		What is you occupation?
		<form>
			<select name="occupation" />
				<option>Teacher</option>
				<option>Nurse</option>
				<option>Priest</option>
			</select>
			<input type="submit" value="Submit" />
		</form>
	</body>
</html>
```



For practical purposes, we have only included three occupations, but the concept is that ideally, one would have a list of all occupations; it would be known what code belongs to each item and once selected there is no interpretation needed by an external process. Unfortunately, a complete list of occupations is too long to fit a web browsers interface, furthermore the codes above are not as fine-grained as researchers might want them to be,

and there are multiple levels and possibly industries where these occupations occur. One approach is to take a list that fully covers all occupations but only on a high level.

For example the ISCO08 level 1:

- Managers
- Professionals
- Technicians and Associate Professionals
- Clerical Support Workers
- Services and Sales Workers
- Skilled Agricultural, Forestry and Fishery Workers
- Craft and Related Trades Workers
- Plant and Machine Operators and Assemblers
- Elementary Occupations
- Armed Forces Occupations

This set of 10 groups are the top level groups of ISCO-08. It is hard for many respondents to identify themselves in this, and to assume that respondents with the same job title do so consistently. An alternative set of top level occupation categories is:

- Agriculture, nature, animals, environment
- Care, children, welfare, social work
- Cars, mechanics, technicians, engineers
- Cleaning, housekeeping, garbage, waste
- Clerks, secretaries, post, telephone
- Commercial, shop, buy and sale
- Construction, fittings, housing
- Education, research, training
- Finance, banking, insurance
- Food manufacturing
- Guards, army, police
- Health care, paramedics, laboratory
- Hospitality, tourism, leisure, sports
- HRM, labour intermediary, organisation
- Industrial production, manufacture, metal
- IT, automation, telecommunication
- Language, library, archive, museum
- Legal, administration, inspection, policy adviser
- Marketing, PR, advertising
- Media, graphic, printing, culture, design
- Oil, gas, mining, utilities
- Management, direction
- Transport, logistics, port, airport

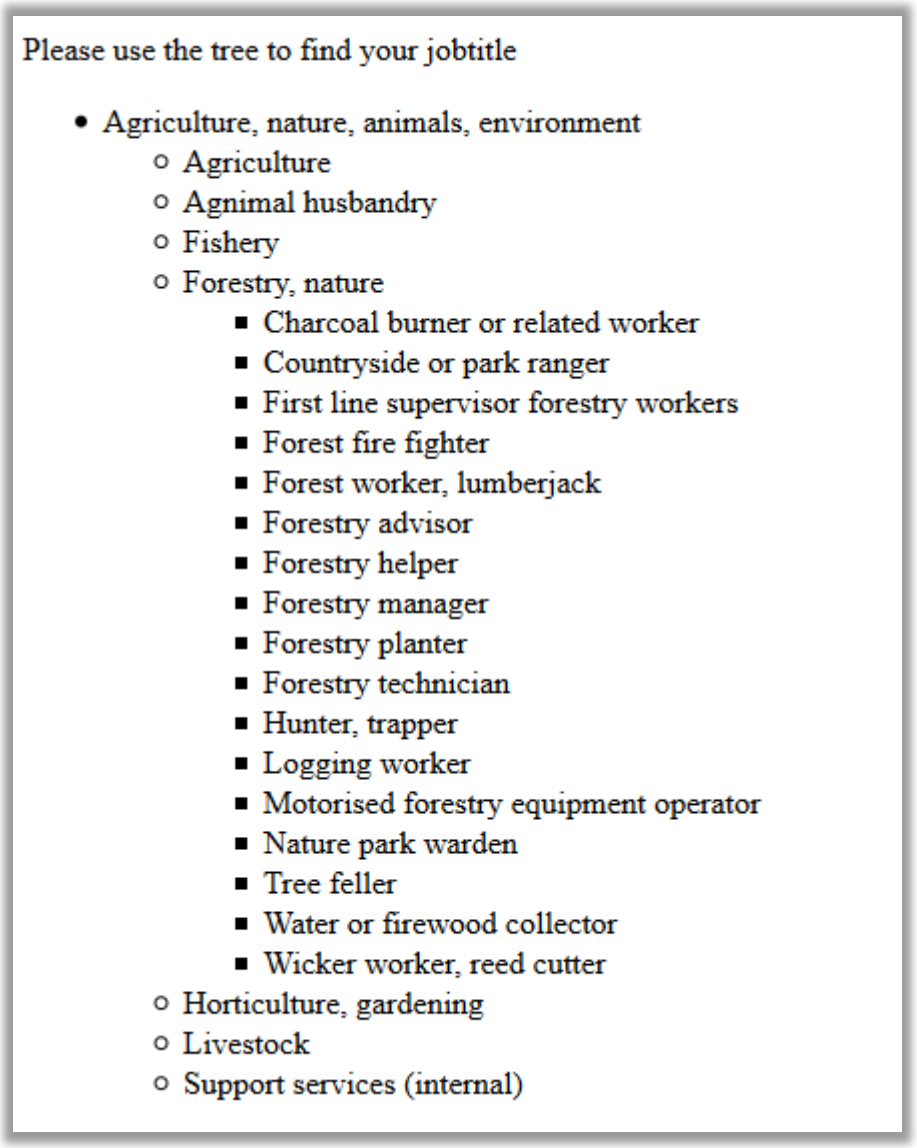
A list like this could be made selectable in html in multiple ways, one of which is by using radio buttons:

```
<html>
    <body>
        What category fits your occupation best?
        <form>
            <input type="radio" value="1">Agriculture, nature, animals, environment
            <input type="radio" value="2">Care, children, welfare, social work
            <input type="radio" value="3">Cars, mechanics, technicians, engineers
            <input type="radio" value="4">Cleaning, housekeeping, garbage, waste
            <input type="radio" value="5">Clerks, secretaries, post, telephone
            <input type="radio" value="6">Commercial, shop, buy and sale
            <input type="radio" value="7">Construction, fittings, housing
            <input type="radio" value="8">Education, research, training
            <input type="radio" value="9">Finance, banking, insurance
            <input type="radio" value="10">Food manufacturing
            <input type="radio" value="11">Guards, army, police
            <input type="radio" value="12">Health care, paramedics, laboratory
            <input type="radio" value="13">Hospitality, tourism, leisure, sports
            <input type="radio" value="14">HRM, labour intermediary, organisation
            <input type="radio" value="15">Industrial production, manufacture, metal
            <input type="radio" value="17">IT, automation, telecommunication
            <input type="radio" value="18">Language, library, archive, museum
            <input type="radio" value="19">Legal, administration, inspection, policy ad
            <input type="radio" value="20">Marketing, PR, advertising
            <input type="radio" value="21">Media, graphic, printing, culture, design
            <input type="radio" value="22">Oil, gas, mining, utilities
            <input type="radio" value="23">Management, direction
            <input type="radio" value="24">Transport, logistics, port, airport
            <input type="submit" value="Submit" />
        </form>
    </body>
</html>
```

These areas of jobs by themselves don't tell us everything we need to identify ISCO levels. They are however good indicators for the job titles. They make it easier for the respondent to self-identify where their job is situated. We use these categories as first level in a tree structure to order the job titles and make them easier to find.

There is no native html input control that shows trees. This has to be constructed. One common way of doing so is by using unordered lists of unordered lists, and format that using JavaScript and styles to look and behave like a dynamic tree.

A branch of such a tree could look like this:



Let's assume respondents who cut wood for a living. These persons initially select a top level "Agriculture, nature, animals, environment". On the next level they select "Forestry, nature", and in the remaining 17 job titles they select the one that comes closest to their occupation, something like "Forest worker, lumberjack".

This example shows only one branch. The complete tree with all branches expanded would become way too big for proper display in this document. The html code of this little example:

```
<html>
	<body>
		Please use the tree to find your jobtitle
		<ul>
			<li>
				Agriculture, nature, animals, environment
					<ul>
						<li>Agriculture</li>
						<li>Animal husbandry</li>
						<li>Fishery</li>
						<li>Forestry, nature
							<ul>
								<li>Charcoal burner or related worker</
								<li>Countryside or park ranger</li>
								<li>First line supervisor forestry workers
								<li>Forest fire fighter</li>
								<li>Forest worker, lumberjack</li>
								<li>Forestry advisor</li>
								<li>Forestry helper</li>
								<li>Forestry manager</li>
								<li>Forestry planter</li>
								<li>Forestry technician</li>
								<li>Hunter, trapper</li>
								<li>Logging worker</li>
								<li>Motorised forestry equipment opera
								<li>Nature park warden</li>
								<li>Tree feller</li>
								<li>Water or firewood collector</li>
								<li>Wicker worker, reed cutter</li>
							</ul>
						</li>
						<li>Horticulture, gardening</li>
						<li>Livestock</li>
						<li>Support services (internal)</li>
					</ul>

			</li>
	</body>
</html>
```

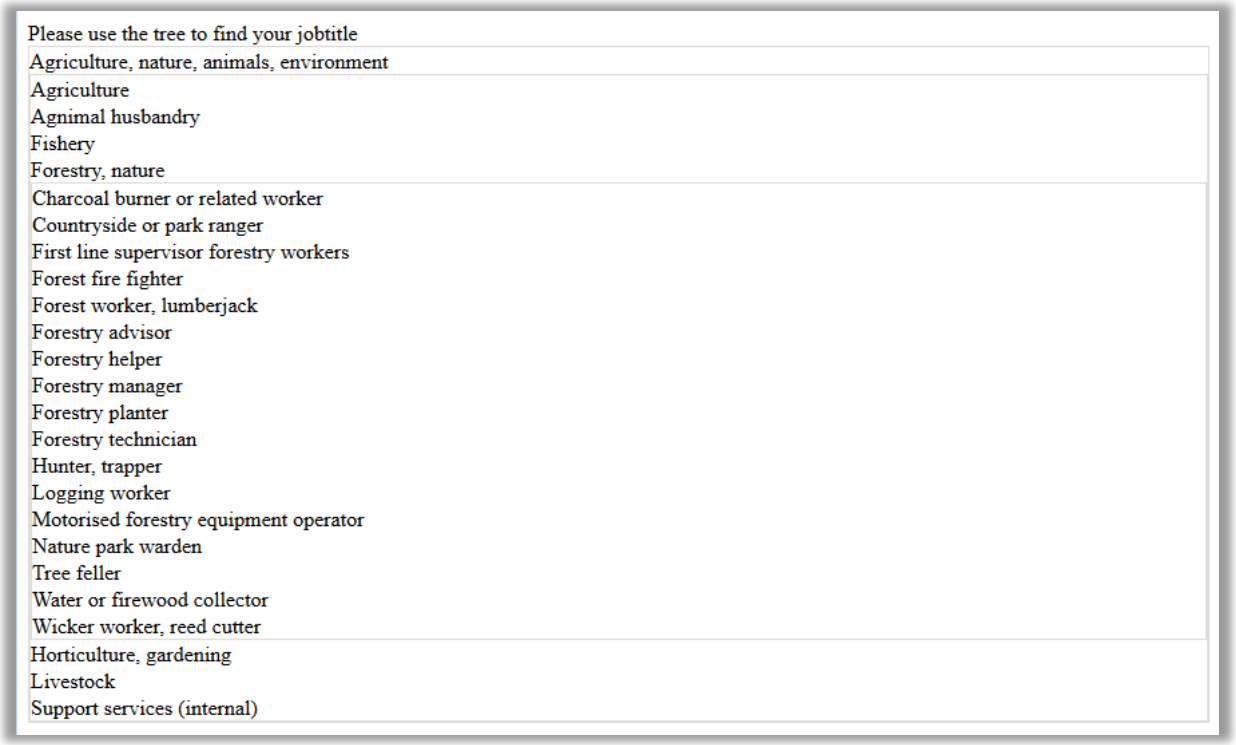
But this code is not dynamic. Ideally once respondents click on an item, the children belonging to that node in the tree would expand and once they select a different node, the previous node would close, to keep the size of the tree limited.

You need to further specify how you want this to look to make it properly fit in your web questionnaires' environment. This can be done using styles.

```
<html>
	<style>
ul {
	list-style: none;
	margin:0;
	cursor: pointer;
	text-align:left;
	border-style: solid;
	 border-width: 1px;
	border-color:#ddd;
	padding:0;
}
	</style>
	<body>
		Please use the tree to find your jobtitle
		<ul>
			<li>
				Agriculture, nature, animals, environment
				<ul>
					<li>Agriculture</li>
					<li>Agnimal husbandry</li>
					<li>Fishery</li>
					<li>Forestry, nature
						<ul>
							<li>Charcoal burner or related worker</
							<li>Countryside or park ranger</li>
							<li>First line supervisor forestry workers
							<li>Forest fire fighter</li>
							<li>Forest worker, lumberjack</li>
							<li>Forestry advisor</li>
							<li>Forestry helper</li>
						</ul>
					</li>
					<li>Horticulture, gardening</li>
					<li>Livestock</li>
					<li>Support services (internal)</li>
				</ul>

			</li>
	</body>
</html>
```

Using the <script> tag we introduced a style on the ul-elements. This changes the tree to display like:



We basically removed the bullets and placed the ul's in boxes. If we now add some styling to the -tags:

```
li {
        background: #EEE;
        line-height:140%;
        text-indent:0px;
        background-position: left top;
        padding-left: 20px;
        background-repeat: no-repeat;
        font-family: Arial, Helvetica, sans-serif;
        cursor: pointer;
}
```

This will make the background grey, change the font, and using the padding it will move the list items slightly to the right:

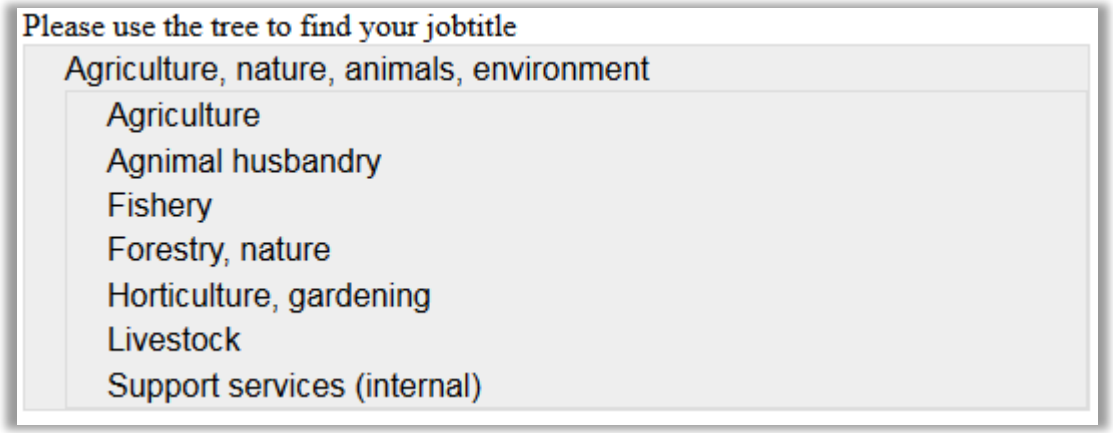
If we want to hide the children of 'Forestry, nature', we have to let the browser know the <ul> within the <li> belonging to "Forestry, nature" should be hidden. To uniquely identify this <ul> you can give it an id:

```
<li>Forestry, nature
        <ul id="children">
                <li>Charcoal burner or related worker</li>
                <li>Countryside or park ranger</li>
                <li>First line supervisor forestry workers</li>
                <li>Forest fire fighter</li>
                <li>Forest worker, lumberjack</li>
                <li>Forestry advisor</li>
                <li>Forestry helper</li>
        </ul>
</li>
```

Them adding a style to 's with id "children":

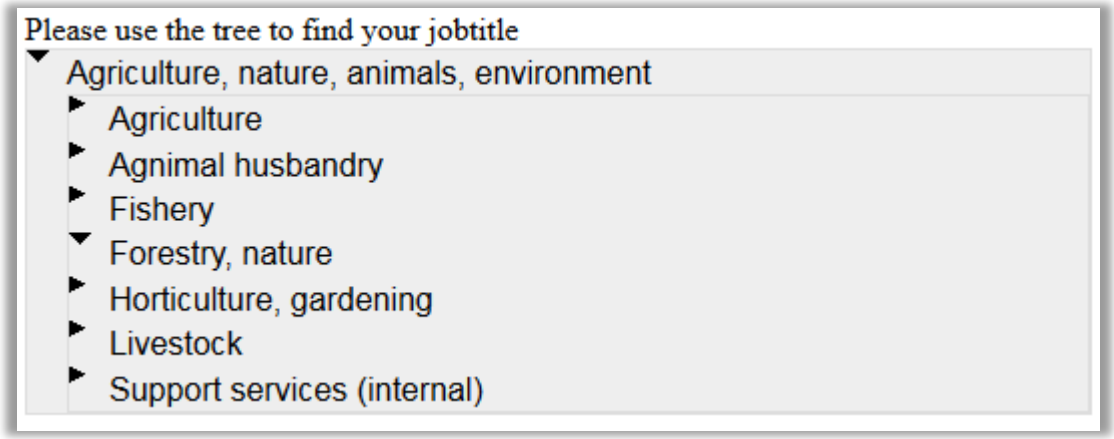
```
ul#children {
        display:none;
 }
```

will hide this block. This is only display, the list is still available, it only doesn't show



We will assign classes ‘collapsed’ and ‘expanded’ to the ’s. Classes are much like id’s, in this respect. An id is used to uniquely identify an element, a class is used to assign elements a certain type. Now if we will add two images, ▼ and ▶ attach them to li’s with class collapsed or expanded:

```
li.collapsed {
   background-image: url(collapsed.png);
}
li.expanded {
   background-image: url(expanded.png);
}
```



The final part of making this tree-view work is making it dynamic. If you click on an item that has children the tree should expand. If you click on another item, the previous expanded one on the same depth should close and a new one should be expanded. This is done using JavaScript.

```
$('#treesuggestions').on("click", "li", function (e) {
    e.stopPropagation();
    $('#treesuggestions').find("ul").hide();

    if (this.className == 'collapsed') {
        $('#treesuggestions').find("li.expanded").attr('class', 'collapsed');
        $(this).attr('class', 'expanded');
        $(this).children('ul').show();
    } else {
        if (this.className == 'expanded') {
            $('#treesuggestions').find("li.expanded").attr('class', 'collapsed');
            $(this).attr('class', 'collapsed');
            $(this).children('ul').hide();
        }
    }

    $(this).parentsUntil($('#treesuggestions')).attr('class', 'expanded');
    $(this).parents().show();
});
```

This code attaches behaviour to the <li> items belonging to an element with id 'treesuggestions'. When you click on such an item it executes the code. First e.stopPropagation is set, to prevent the click event to also activate items within the <li>. All ul's are hidden. We want to toggle the display between collapsed and expanded. So if the list item currently has className 'collapsed' it should become 'expanded'. First we collapse any branch that is currently expanded. Next we set the correct className. And show or hide the ul within the li. When the item is set we will make sure the path of parents up to the current list get status expanded and are visible.

Now we have all building block needed to get a dynamic tree. We could design one tree in html-code like above and with some extra code, you can attach whatever item is selected to the input item your questionnaire item generated.

It is a lot of work to maintain and further develop such a tree. Therefore, we think it is better to generate it from the surveycodings service. This service make sure the list is up to date and will be further improved. We chose to use a data structure defined in JSON to do so. JSON means JavaScript Object Notation, it is a structured way of defining the tree.

You could use a file with the tree definition, define it in your code, or call it from an external website.

This is some basic example JQuery code you can use to call the service:

```
$.getJSON(
"https://api.surveycodings.org/codings/search.json?type="+type+"&context="+context_id+"&mode=tree&callback=?", function (res, code) {
        // load results from res and code in the html structure
});
```

To use the service in your code you first have to decide on the string you want to call the service with. The structure of these strings is explained in the previous section. You call the service using some predefined parameters, and the structure of the JSON code is defined. We already decided what the structure of the html should be, so what remain is code transferring the JSON definition into html-structured code.

We use the following code to do so:

```
$.getJSON("https://api.surveycodings.org/codings/search.json?type="+$_GET['type']+"&context=" + context_id + "&mode=tree&callback=?", function (res, code) {

    searchResponse = res.trees;
    sugList.html("");
    var j = 0;
    var color = [];

    for (var i in searchResponse) {
      parent_node_id = 0;
      if (searchResponse[i].parent_node_id != null) {
        parent_node_id = searchResponse[i].parent_node_id;

      }

      if (searchResponse[i].parent_node_id in color) {

        color[parent_node_id] = ((color[parent_node_id] + 1) % 2);

      } else {

        color[parent_node_id] = 0;

      }


      if (parent_node_id == 0) {

        j = color[parent_node_id];

        if (j == 1) {

          sugList.append("<li id=\"" + searchResponse[i].id + "\" test=\"" + j + "\" style=\"background-color:#EEE;\" code=\"" + searchResponse[i].coding_name + "\">" + searchResponse[i].coding_description + "</li>");

        } else {

          sugList.append("<li id=\"" + searchResponse[i].id + "\" test=\"" + j + "\" code=\"" + searchResponse[i].coding_name + "\">" + searchResponse[i].coding_description + "</li>");

        }

      } else {


        if ($("#" + parent_node_id).length > 0) {

          //parent exists

          if ($("#" + parent_node_id + " ul").length > 0) {
```

The jQuery code above will populate a tree structure generated from the surveycodings API.

Still many respondents would find it hard to identify their job using trees. For example, how would an ‘it-specialist in a hospital’ navigate the tree? A mistake on the top level hides the path to the occupational category.

Ideally one would simply type in an occupational title and it would be assigned to the correct classification, which is the right indicator for a respondent’s occupation. Asking a respondent to type in their job title is indeed something that is done often. You can ask in a web questionnaire a respondent to type in the title of their occupation, store this in the data and do (semi-automatic) office coding by a team of experts afterwards, and for many occupations that would be a decent way of finding a suitable classification. However there are a few drawbacks with this approach.

First, human labour is involved in this process, a human has to interpret the string that was entered, has to weigh other information available to assess the response. Part of this issue can be solved by some form of automated coding, based on some rules or statistical approach a good estimate could be made, but a large subset will remain uncodable. This is the second drawback of allowing open text answers. The respondent can type something in that is not codable. A respondent could for instance type in the name of their employer, or the industry they work in. Or possible a too broad title (manager) or possibly too domain specific or a maybe even a job title that is new. For these problematic open answers office coding runs into problems. Ideally you want do a review of the answer immediately after the respondent has answered the question and verify the coded item, see if you can code it, verify if the coded item is likely based on other question responses. When this check fails we could ask for an alternative of more specific item, or possibly ask for more information to better understand the occupation.

But why wait until the respondent has entered in their response? As respondents type in their open answer, we could already try to find potential matches in a list of occupation titles. As respondents type in we show a list of items that could potentially be matches, each time the list with matches could become more specific or broader based on the input. This would give the respondent immediate feedback, if they type in something that can’t be handled by the algorithm there won’t be matches and the respondent could use an alternative wording. If the title is too broad, a longer list of potential matches is shown, indicating that they should further specify or maybe select something from the list.

To support this last process an API has been developed that can be called. To match texts against the database of occupational titles, on the surveycodings.org server, a service is listening. This means, you can type in this address in your browser and you will get back a structured description of your search results.

The url of the service is:

https://api.surveycodings.org/codings/search.json

Calling this url in your browser address bar will give an error. It needs to have parameters that should tell it what it is supposed to do.

Such parameters are *type, mode* and *context*

For example:

https://api.surveycodings.org/codings/search.json?type=occupation&mode=search&context=en_GB&search=pris

Will return:


```
{"codings":{"13440200":"Prison manager","54130100":"Prison
guard","54130200":"First line supervisor prison guards"}}
```


This is a JSON encoded list with codes and occupations that match the search term ‘pris’:

13440200 Prison manager
54130100 Prison guard
54130200 First line supervisor prison guards

If you were to call this service with context fr_FR this list would be different:

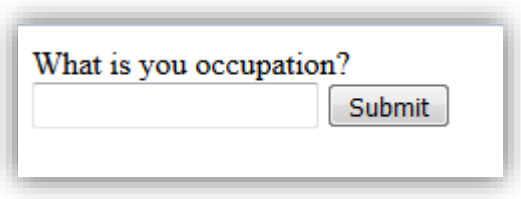

```
{"codings":{"13440200":"Prison manager","54130100":"Prison
guard","54130200":"First line supervisor prison guards"}}
```


Instead of using the mode ‘search’ we could also as for a tree-representation of the French context:

https://api.surveycodings.org/codings/search.json?type=occupation&mode=tree&context=fr_FR

Again we will show more in depth how to set this up in a webpage. We earlier showed html code for an open answer question:

```
<html>
      <body>
            What is you occupation?
            <form>
                  <input name="occupation" type="text" />
                  <input type="submit" value="Submit" />
            </form>
      </body>
</html>
```



We adapted the input type ‘text’ with a ‘search’ and included a hidden context input which could send the locale information to the server. We also removed the Submit button for now and will submit on click in our example.

```
<div id="codingsSelectForm" class="form-group">
        <p>What is your occupation?</p>
        <input id="context" value="en_GB" type="hidden">
        <input class="form-control" id="description" placeholder="Search"
title="Search" autocomplete="off" dir="ltr" type="search">
        <div id="navigation" style="">
                <ul id="suggestions"></ul>
        </div>
</div>
```

If we use similar jQuery code as in the tree example we can call the API and parse the resulting JSON structure into the suggestions .

Calling:
https://api.surveycodings.org/codings/search.json?type=occupation&mode=search&context=en_GB&search=fru

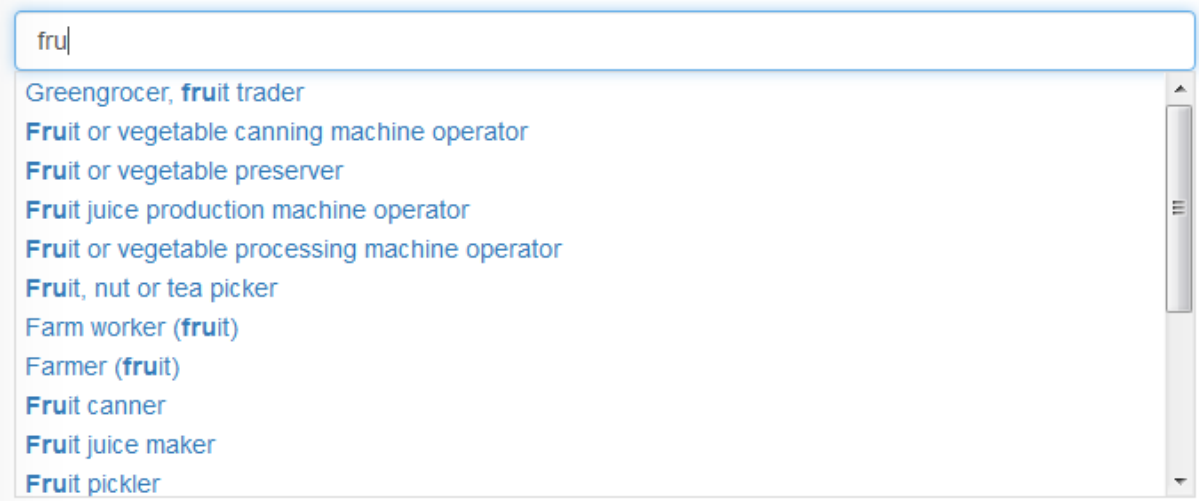
Gives back JSON structure:

```
{"codings":{"5221160000000":"Greengrocer, fruit
trader","7514010000000":"Fruit or vegetable canning machine
operator","7514030000000":"Fruit or vegetable
preserver","8160190000000":"Fruit juice production machine
operator","8160200000000":"Fruit or vegetable processing machine
operator","9211020000000":"Fruit, nut or tea picker","6112000600016":"Farm
worker (fruit)","6112001700016":"Farmer (fruit)","7514000300016":"Fruit
canner","7514000400016":"Fruit juice maker","7514000500016":"Fruit
pickler","7515000200016":"Fruit grader","8160000800016":"Fruit press
operator","8160001500016":"Machine operator (fruit
processing)","9211000300016":"Farm hand (citrus
fruit)","9333000700016":"Fruit porter "}}

This code can easily be transformed into html code:
```

```
<li id="5221160000000" class="unselected">
        <a class="listitems">
                Greengrocer, <b>fru</b>it trader
        </a>
</li>
<li id="7514010000000" class="unselected">
        <a class="listitems">
                <b>Fru</b>it or vegetable canning machine operator
        </a>
</li>
<li id="7514030000000" class="unselected">
        <a class="listitems"><b>Fru</b>it or vegetable preserver</a>
</li>
<li
        ..............
<li id="9333000700016" class="unselected">
        <a class="listitems"><b>Fru</b>it porter</a>
</li>
<li id="-20070" class="unselected">
        <a class="listitems"><b>Fru</b>it</a>
</li>
```

Which will display:



If it is not possible to call the surveycodings.org service, survey holders may want to set up their own service. To do so, they should hire server capacity and build a service themselves or get in contact with CentERdata to acquire some sample source code which then needs further tailoring to the local environment.

If you rather not use a service at all, but simply want to include the complete list of occupations/industries/occupations on your website, this is possible too. One approach with this is to walk through the list and do some text matching and only show the items that match similar to the code above. We found however that as the lists becomes larger the script becomes very slow. A trick we developed to make the matching of downloaded JSON list faster is by applying text matching directly on the JSON string using a regular expression making the JSON string shorter before displaying the list.

You could define a text variable which contains the complete codings list in JSON format. The structure will be the same as in the example, the filtering of the results should be done on the client side.

A code snippet describing this:

```
$("#answerfield").on("input", function (e) {
        var searchKeywords = $(this).val();
        if (searchKeywords != "") {
                //if there is a search string, run search and update list
                searchKeywords = escapeRegExp(searchKeywords);
                searchItems = searchKeywords.split(" ");
                var codes_search = codes;
                codes_search = codes_search.replace("]",",]");

                for (var i=0; i<searchItems.length; i++) {
                        if(searchItems[i]!="") {
                        var pattern = new RegExp('\{"code\"\:\"[A-Za-z0-
                        9]+\","description"\:"((?!("|' + searchItems[i] + ')).)*"\},','ig');
                        codes_search = codes_search.replace(pattern, "");
                    }
                }

                codes_search = codes_search.replace("[,","[");
                codes_search = codes_search.replace(",]","]");
                fillList(codes_search,  selected_code);
        } else {
                //if search is empty, show all
                 fillList(codes, selected_code);
        }
});
```

This will filter out titles with no matching substring form the JSON structure. This is a very crude mechanism, but will work way quicker than a case by case analysis.

4. How can survey holders use the API in CAPI?

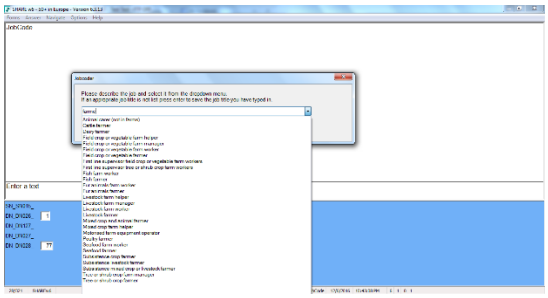
CAPI mode stands for Computer Aided Personal Interviewing. This means that an interviewer conducts an interview using a computer. It is common practice that an interviewer meets the respondent at the respondent's house and has the interview there.

These days it is not yet common to have an internet connection available during such an interview. Typically, the interviewer uses a laptop to fill in the answers a respondent gives.

Since we can't assume connectivity, it is not possible to use the service at surveycodings.org. We should think of alternative strategies to use the validated list of socio economic indicators.

There are many software tools available that run interviews on laptops so it is not possible to present one solution that will fit all. Below we will describe some strategies on how to include our lists in your system.

We see 5 strategies:

- Use a tool, that can be called from the interview software; in some interviewer software tools it is possible to call an external library that behaves like a service, you can call functions and get responses back.
- Create a tool, which can be triggered from the interview software; for example the software as used in ShareW6 works like this. An explanation of this tool can be found in the methodology book of Share wave 6: Malter, F. and A. Börsch-Supan (Eds.) (2017). SHARE Wave 6: Panel innovations and collecting Dried Blood Spots. Munich: Munich Center for the Economics of Aging (MEA). (See: http://www.share-project.org/fileadmin/pdf_documentation/2017-12_SHARE_Wave6_MFRB.pdf)
- 
  Load the lists in your survey software; if your survey software system allows you to setup databases or preload data, it could be a good strategy to load these datasets into the system. You can the build a lookup table on this.
- Run a web service on your own computer and let the survey software make calls to this. Some newer survey software tools actually use internet technology to run CAPI tools. This opens the possibility to run a service on the laptop. You could use the same calls and scripts as described in the web modes.
- A last resort would be to program a web layer over your CAPI software as was done for Share Wave7 as described in the paper 'A web compatible Dep' by Maurice Martens (see http://www.blaiseusers.org/2016/papers/3_5.pdf) which was published in the Complete Volume of the 17th IBUC 2016 and presented at the International Blaise User Conference 2016 in The Hague, the Netherlands.

We have some tools available that work with some surveying packages, but we expect that some solutions will need to be tailored to fit.

To use the verified lists on tablets or smartphones, things do not really differ that much from CAPI unless you know there is a connection with the internet.

What the best strategy is for tablets, depends on whether you have (or can assume) a connection. It is harder to call outside programs from your app, and many apps currently are somewhat limited in the accessibility of loading lookup tables.

5. References

European Commission, Directorate-General For Research & Innovation, Research infrastructure (2015) ANNEX 1 (part A) Research and Innovation action NUMBER — 654221 — SERISS, Brussels

Tijdens, K.G. (2016a) Database of industries + explanatory note. Deliverable D8.10 of the SERISS project funded under the European Union's Horizon 2020 research and innovation programme GA No: 654221. Available at: www.seriss.eu/resources/deliverables

Tijdens, K.G. (2016b) Survey Q&A + explanatory note. Deliverable D8.14 of the SERISS project funded under the European Union's Horizon 2020 research and innovation programme GA No: 654221. Available at: www.seriss.eu/resources/deliverables .