JSON Demo Configurations

Concept:

The purpose of JSON demo Configurations are to make demos easier and quicker to produce. The demo builder will send a JSON configuration string from the server and will build a demo based on it. Besides the decreased time in production these configurations will be coded for maximum efficiency on mobile devices. More aesthetically the templates will also help make the demos look consistent. Templates can be used to make completely JS free demos or can be adding to custom demos. For an example of adding a template to a custom demo see I*ncluding a template question in a custom demo (for developers)* below.

Research:

JSON configurations were built so that someone with no programming experience could create them. That being said it would be beneficial to research a bit before attempting to create a configuration:

http://www.json.org/ - The main website explaining what JSON is.
http://json.org/example.html - Some examples of JSON data.

Tools:

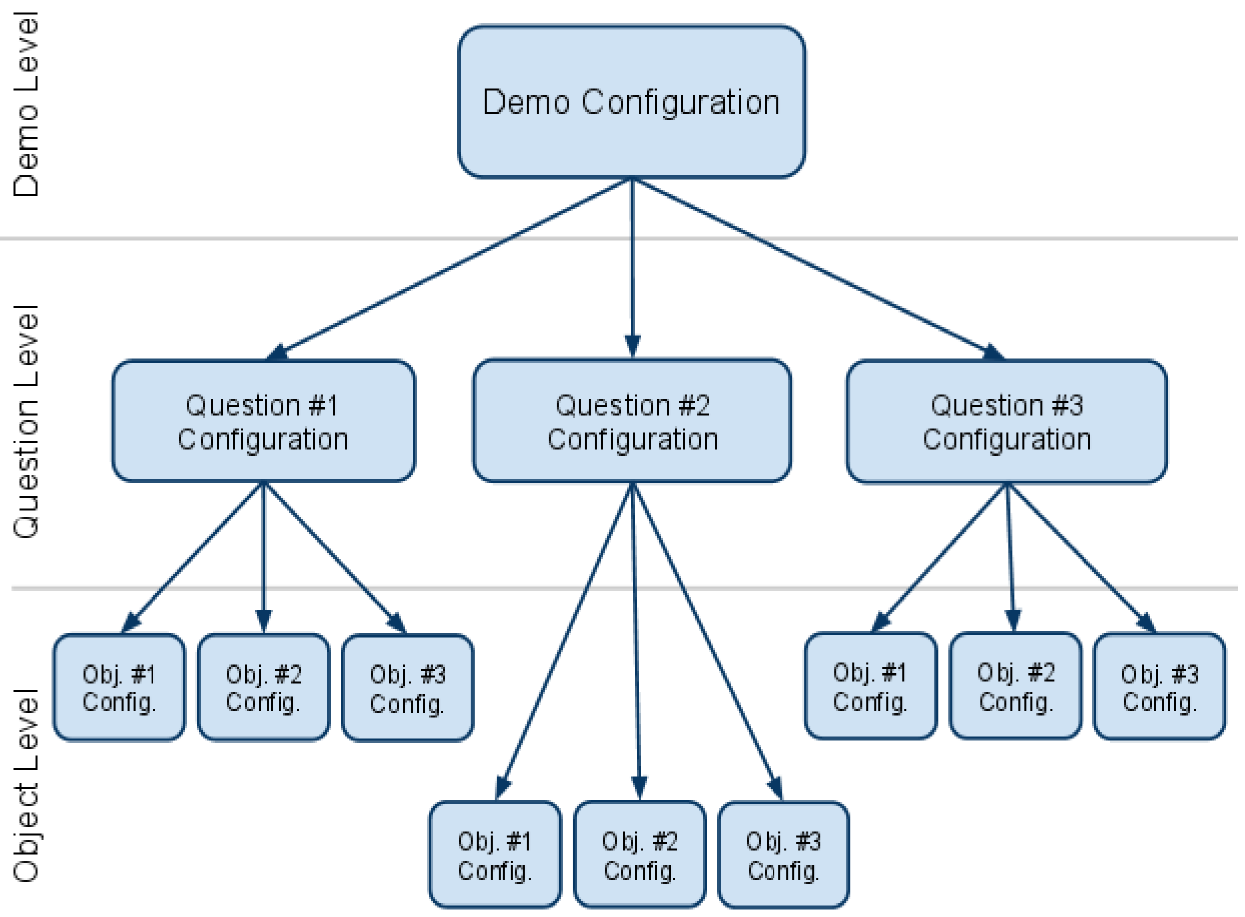
To start creating configurations all you really need is a text editor like notepad. It would be beneficial to have a text editor with context sensing and good tab alignment. This list is not the be all and end all of what you might need but it's a good place to start:

http://www.colorpicker.com/ - A program to help generate HTML color codes.
http://notepad-plus-plus.org/ - A great text editor for windows.
http://www.barebones.com/products/textwrangler/ - An ok text editor for Mac OSX.
http://projects.gnome.org/gedit/ - A good text editor for linux and other BSD systems.
http://www.dropbox.com/ - A useful online data storage tool that can host images.

JSON organization:

Every JSON object is organized in a tree structure. At the base of the tree are the overall demo configurations such as demo title, demo instructions the media URL location and the unique ID slug. The next level of the tree is the question array that contains configurations for every question in the demo. This typically includes the question type, question text and the background image used. The question level will have more configurations based on the question type selected. The top level of a JSON object is the object level which contains configuration for each object. This typically includes the background color, rounded corners, images and text.

Refer to the below sketch of a 3 question demo with 3 objects in each question. Because of the flexible natures of trees you can have pretty much any number questions and objects. The limiting factor is screen real estate and memory.



Defining strings and arrays

Each configuration parameter is set using a JSON style assignment. At the beginning is always curly open bracket “{“ then the name of the parameter follow by a full colon “:” then a string or an array which is the parameters value. A string is a list of alpha-numeric characters “strung” together to form words and sentences. They are surrounded by single or double quotation marks which must be consistent (ie. if you start with double quotation marks you must end with double quotation marks). Note: if you need to use quotation marks inside a string prefix a backslash in front for example: “\”Nevermore\” said the raven.”.

Arrays are indexed lists of values. They can be any type of data but in these configurations they will always be a string or another array. Arrays are started with a square open bracket “[“ followed by the first element then a comma. Each following element is comma separated until you reach the end of the array which is capped off with an square close bracket “]”. As an example here’s an array of five strings [“one”,”two”,”three”,”four”,”five”]. Arrays can be any size and is more limited my system memory.

At the end of a JSON assignment you can use a comma if you’re going to added another JSON assign after or capped it off with a curly close bracket “}”. Although is not necessary it would be much easier to tab indent each level in a JSON configuration. Consider the following two JSON examples that contain the exact same information:

Method #1

```
{
	title:"Demo Level",
	question:
	[{
		title:"Question #1",
		objects:
		[{
			title:”Object #1”,
		},{
			title:”Object #2”
		}]
	},{
		title:"Question #1",
		objects:
		[{
			title:”Object #1”,
		},{
			title:”Object #2”
		}]
	}]
}
```

Method #2
{title:"Demo Level",question:[{title:"Question #1",objects:[{title:”Object #1”,},{ title:”Object #2”}]},{title:"Question #1",objects:[{title:”Object #1”,},{title:”Object #2”}]}]}

Even though both configurations would have the same effect the first method is significantly easier to read and edit. The computer can read second method as easily as the first but a human would really struggle with the second method. Once a configuration has been uploaded to the server it will undergo some heavy compression and will end up looking like the second method. Because of this there is no need to condense the JSON configuration before hand.

HEX Color codes

All color attributes inside JSON configurations are in either 24bit or 32bit hexadecimal. This is the typical method of color encoding for any web development. Since there many websites that do a great job of explaining this there is no need to cover it here. HTML Goodies has a really article on HTML color code here:

http://www.htmlgoodies.com/tutorials/colors/article.php/3478951/A-Quick-Color-Explanation.htm

On major deviation from HTML color codes is in HTML you are only able to assign 24bit colors. In hex this translates to a 6 digit hexadecimal code. For the following example we’ll assume the hex code “9933ff” which translates to purple. Since the plugin works in 32bit mode this allows us to define a color with varying opacity. So to define the same color purple with 50% opacity you would add an extra two digits “80” to the end making the color “9933ff80”. As a note colors needs to be either 6 of 8 digits long or the computer will misinterpret them.

Console Log

The demo builder will output status information while it’s building the demo. This hidden from the user by default and if the demo runs into an error it will continue the best it can. If you run into JSON configuration that doesn’t output what you expect it would be worth going through the build log. The log is verbose and the builder will log any instance that it couldn’t find a configuration value.

To view the console you use either Chrome or Firefox. With Firefox you’ll have to download an extension call Firebug (http://getfirebug.com/). Then after the demo has ran open up Firebug and click of the console panel. Chrome come with the console integrated. It can be activated by right clicking on an empty spot of the webpage and select the option “Inspect element”. Then to view the log select the console from the fire bug panel.

Demo Configuration

Demo configuration parameters are at the top level of settings. Typically these configurations affect the entire demo. For all configurations options if the parameter is omitted a smart default is substituted. The demo well also print an error that it couldn’t find a parameter. This error message is more for debugging purposes in case a parameter doesn’t seem to be taking effect.

Parameter	**Default**	**Description**
title	"demo title"	The display title of the demo located at the top right of the screen.
instructions	"demo instructions"	The displayed instructions that pops up once at the beginning of a demo.
mediaURL	""	The URL location of all the graphics used in the demo. When first creating a configuration this will probably point to a dropbox then once the demo is live it will point to S3.
slug	""	To prevent name conflicts we put a UUID slug in front of every image once it’s put on S3. The slug is generated off of the title once the demo goes live.
questions	required	This is an array of all the questions in the demo. Each question contains it’s own JSON configuration.

General Question Configuration

Question configuration parameters are at the middle level of settings. Typically these configurations affect the only the question they detail.

Parameter	**Default**	**Description**
text	"Question text"	The display text of the question located at the top of the screen.
x	"20"	This is where the question content starts horizontally.

y	"20"	This is where the question content starts vertically.
width	“440”	This is how long the question content is horizontally.
height	“220”	This is how the tall the question content is vertically.
image	“”	This is the background image displayed behind the question content. (Remember most web servers are case sensitive.)
type	“ordering”	The type of the question will dictate which template is use for the question. It will also affect what parameters the demo builder looks for at the object level.

Ordering Configuration

For ordering questions each object is arranged into an even grid and the user can shift around the order of the objects. The order that the objects are defined is the correct answer that the template is looking for. The template will attempt to randomize the given order up to 10 times and will log an error if it fails for whatever reason (usually not enough objects defined). This type of demo works best for 3 to 10 items per question.

Ordering Specific JSON:

Parameter	**Default**	**Description**
objects	required	These JSON objects define what each ordering item looks like. (*see Ordering Objects JSON below*)
layout	“vertical”	The layout affects weather the demo orientates itself to be “vertical” or “horizontal”.

Ordering Objects JSON:

Parameter	**Default**	**Description**
background_color	"FFCC99"	The color of the background rectangle drawn at the bottom of the object. (Note if you don’t want any background color then use “00000000”)
rounded_corners	10	Rounds the corner of the rectangle by the radius of the parameter in pixels.
text	“”	The text displayed inside the object.
text_align	“center”	The alignment of the text which can be “left”, ”center” or ”right”.
text_color	“000000”	The color of the text.
text_size	12	The size of the text in points.
text_layout	“full”	How the object layout is setup. (*see object layouts below for additional information*)
image	“”	The image drawn inside the object.
image_width	full width	The width of the image. Defaults to the full height of the object.
image_height	full height	The height of the image. Defaults to the full height

		of the object.
vertical_padding	5	The number of pixels used as a vertical border.
horizontal_padding	5	The number of pixels used as a horizontal border.

Matching Configuration

For ordering questions each object is separated into an A and B group. The number of objects in each group does not need to match. The user will make a connection by dragging a line from an object in one group to an object in another group. The question will remove any conflicting connections and a user can remove a connection by clicking on one it's objects. The minimum size each group should be three and the maximum is only limited by the screen size.

Ordering Specific JSON:

Parameter	**Default**	**Description**
group_a	required	These JSON objects define the objects on the left/ bottom side. (*see Matching Objects JSON below*)
group_b	required	These JSON objects define the objects on the right/ top side. (*see Matching Objects JSON below*)
pairs	required	The pairing answers for this question (*see Pairs JSON below*)
layout	"vertical"	The layout affects weather the demo orientates itself to be "vertical" or "horizontal".
disabled_color	"c0c0c0"	The color an object background changes to the visually cue the user they been connected.
overlay_color	"00000080"	The color of the line that is dragged around.
connection_color	"000000a0"	The color of the connection line drawn in between to connected objects/

Matching Objects JSON:

Parameter	**Default**	**Description**
name	required	The name the question uses when trying to link together a pair. This is require since the answer pairs reference this value.
background_color	"FFCC99"	The color of the background rectangle drawn at the bottom of the object. (Note if you don't want any background color then use "00000000")
rounded_corners	10	Rounds the corner of the rectangle by the radius of the parameter in pixels.

text	“”	The text displayed inside the object.
text_align	“center”	The alignment of the text which can be “left”, ”center” or ”right”.
text_color	“000000”	The color of the text.
text_size	12	The size of the text in points.
text_layout	“full”	How the object layout is setup. (*see object layouts below for additional information*)
image	“”	The image drawn inside the object.
image_width	full width	The width of the image. Defaults to the full height of the object.
image_height	full height	The height of the image. Defaults to the full height of the object.
vertical_padding	5	The number of pixels used as a vertical border.
horizontal_padding	5	The number of pixels used as a horizontal border.

Pairs JSON:

Parameter	**Default**	**Description**
reference_a	required	The name of the first object in this pair.
reference_b	required	The name of the second object in this pair.

Placement Configuration

For placement questions each object is arranged into a stack and the user can position the object by placing them into a predefined socket. The socket keeps track of which object has been placed inside it and which object is the correct object. When defining the stack object name and the socket correct object name make sure then are identical including the case and punctuation. The template will log an error if it can’t find matching names. The template will attempt to randomize the stack up to 10 times and will log an error if it fails for whatever reason (usually not enough objects defined). This type of demo works best for 3 to 10 items per question.

Placement Specific JSON:

Parameter	**Default**	**Description**
stack_x	20	The X location where the stack will start.
stack_y	20	The Y location where the stack will start.
stack_offset_x	4	The amount in the X direction each object in the stack will be offset by.
stack_offset_y	4	The amount in the Y direction each object in the stack will be offset by.
max_distance	75	The maximum distance in pixels that an object from the stack will consider snapping to a socket.
stack	required	The list of each object in the stack. (*see Stack JSON below*)
sockets	required	The list of each socket and the correct answers. (*see Socket JSON below*)

Stack JSON:

Parameter	**Default**	**Description**
background_color	"FFCC99"	The color of the background rectangle drawn at the bottom of the object. (Note if you don’t want any background color then use “00000000”)
rounded_corners	10	Rounds the corner of the rectangle by the radius of the parameter in pixels.
text	“”	The text displayed inside the object.

text_align	“center”	The alignment of the text which can be “left”, ”center” or ”right”.
text_color	“000000”	The color of the text.
text_size	12	The size of the text in points.
text_layout	“full”	How the object layout is setup. (*see object layouts below for additional information*)
image	“”	The image drawn inside the object.
image_width	full width	The width of the image. Defaults to the full height of the object.
image_height	full height	The height of the image. Defaults to the full height of the object.
vertical_padding	5	The number of pixels used as a vertical border.
horizontal_padding	5	The number of pixels used as a horizontal border.
name	“untitled”	The name of the object, use for checking of the answer is correct.
width	“128”	The total width of the stack object
height	“32”	The total height of the stack object

Sockets JSON:

Parameter	**Default**	**Description**
image	“”	The image drawn inside the object.
image_width	”128”	The width of the image.
image_height	“128”	The height of the image.
name	“untitled”	The name of the object, use for checking if the answer is correct.
point_x	“100”	The x-axis center point of the socket.
point_y	“100”	The y-axis center point of the socket.

Labeling Configuration

For labeling questions the relationships and arrows must be created before hand on an image. If the is smaller then the layout width and height then the is willing be draggable along with all the labels attached to it. Labeling questions introduces the concept of question inheritance which allows multiple questions to share resources and be linked together. A great example of this is the Symbolic Interactionism demo which builds on top of each previous question.

The most important concept for question inheritance is fact that their is a parent and one or more child questions. The parent question should be the first question in the series and will define the look and feel of child questions. The parent question must be named since that's the method the configuration uses to find a parent internally. When a child inherits from a parent it links itself to the parent so it can share resources. In the of parameters below the description will indicate if the parameter is inheritable. Besides the fact this method allows you to link questions together it also reduces the amount of parameters needed in a child question.

Labeling Specific JSON:

Parameter	**Default**	**Description**
name	required if parent	The name used to a child question looks for to link to a parent question.
inherit	required if child	The parent name the child will use to link to the parent question.
x	"20"	This is where the question content starts horizontally. (*Inherits parameter from parent*)
y	"20"	This is where the question content starts vertically. (*Inherits parameter from parent*)
width	“440”	This is how long the question content is horizontally. (*Inherits parameter from parent*)
height	“220”	This is how the tall the question content is vertically. (*Inherits parameter from parent*)
text_height	“50”	The height of the semi-transparent background for easing the reading of the questions text.
image	“”	This is the background image displayed behind the question content. (*Inherits parameter from parent*)

image_width	“480”	The width of the background image. (*Inherits parameter from parent*)
image_height	“320”	The height of the background image. (*Inherits parameter from parent*)
tween_x	“20”	The x position the background image tweens to when the question is displayed. This allow you to answer concept maps questions in sections.
tween_y	“20”	The y position the background image tweens to when the question is displayed. This allow you to answer concept maps questions in sections.
labels	required	Each questions defines an active group on labels for that question. When a child question is linked to a parent question it’s labels are appended to the parent.

Labeling JSON:

Parameter	**Default**	**Description**
x	"0"	This is where the drop down/text box menu starts horizontally based on the background image.
y	"0"	This is where the drop down/text box menu starts vertically based on the background image.
width	“80”	This is how long the drop down/text box menu is horizontally.
height	“20”	This is how the tall the drop down/text box menu vertically when expanded. When closed the drop down menu is ~20 pixels tall.
answer	“”	This the answer for the concept. The answer must match of the options or the question will be impossible to answer.
tolerance	“0”	The acceptable range of a numeric answer. A users answer will be considered correct when answer +/- tolerance.
options	[“”]	The options is an array of strings to that are listed in the drop down menu. *Note: if no options are defined then the question will use a text box*

		instead of a drop down.

Click on target configuration

Click on target questions can inherited in a similar fashion as labeling questions. A target has three states images: enabled, disabled and hit. Enabled is the state that targets start in and is used when the target is not selected. Disabled is the state that a targets is set to when it's not active and can't be used in the the current question. Hit is the state the target has been clicked on and is selected.

Click on target Specific JSON:

Parameter	**Default**	**Description**
name	required if parent	The name used to a child question looks for to link to a parent question.
inherit	required if child	The parent name the child will use to link to the parent question.
x	"20"	This is where the question content starts horizontally. (*Inherits parameter from parent*)
y	"20"	This is where the question content starts vertically. (*Inherits parameter from parent*)
width	“440”	This is how long the question content is horizontally. (*Inherits parameter from parent*)
height	“220”	This is how the tall the question content is vertically. (*Inherits parameter from parent*)
text_height	“50”	The height of the semi-transparent background for easing the reading of the questions text.
image	“”	This is the background image displayed behind the question content. (*Inherits parameter from parent*)
image_width	“480”	The width of the background image. (*Inherits parameter from parent*)
image_height	“320”	The height of the background image. (*Inherits parameter from parent*)

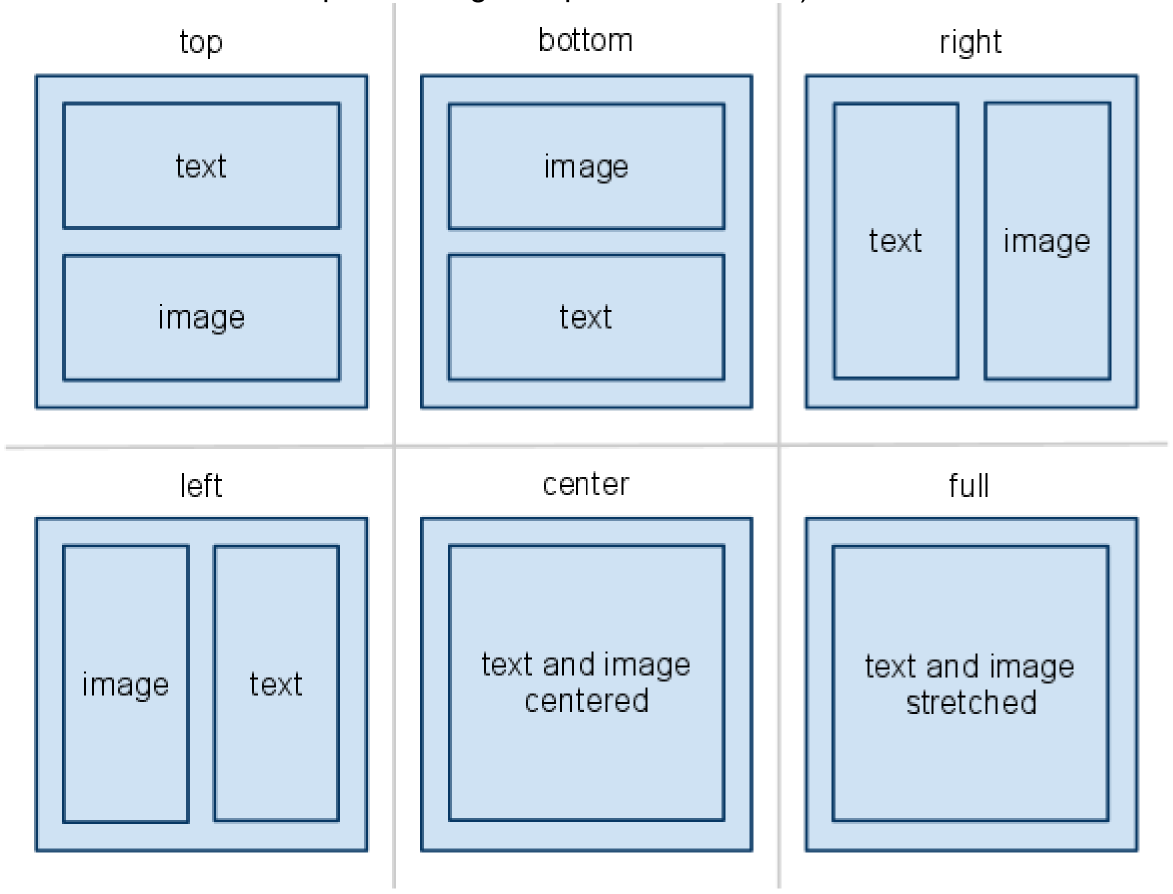
tween_x	“20”	The x position the background image tweens to when the question is displayed. This allow you to answer concept maps questions in sections.
tween_y	“20”	The y position the background image tweens to when the question is displayed. This allow you to answer concept maps questions in sections.
multiple_answers	“false”	If true the question allows more then one target to be selected at once.
targets	required	Each questions defines an active group on targets for that question. When a child question is linked to a parent question it’s targets are appended to the parent.

Targets JSON:

Parameter	**Default**	**Description**
x	"0"	This is where the drop down/text box menu starts horizontally based on the background image.
y	"0"	This is where the drop down/text box menu starts vertically based on the background image.
width	“32”	This is how long the drop down/text box menu is horizontally.
height	“32”	This is how the tall the drop down/text box menu vertically when expanded. When closed the drop down menu is ~20 pixels tall.
imageEnabled	“”	The URL of the enabled image
imageDisabled	“”	The URL of the disabled image
imageHit	“”	The URL of the hit image
correct	“false”	If true then the target must be selected and if false the target must be unselected.

Object Layouts

Each ordering object can have it's own layout. The valid layouts are (“left”, ”right”, ”top”, ”bottom”, “full” and “center”). The below diagram shows what the object will look like depending on the layout. (Note the difference between “full” and “center” is “full” stretches the image the full size of the object regardless of image size while “center” keeps the images aspect ratio in tact.)



Including a template question in a custom demo (for developers)

A template question can be included into a custom demo with ease. The file THP_Config.js will need to be included inside the demo and then all the template question functions will be available to you. Each template is an extended scene with that builds itself depending on the JSON passed in. The follow templates are available:

THM_OrderingQuestion (plugin, configuration)
THM_MatchingQuestion (plugin, configuration)
THM_PlacementQuestion (plugin, configuration)
THM_ClickOnTargetQuestion (plugin, configuration, thmDemo)
THM_LabelingQuestion (plugin, configuration, thmDemo)

These functions take two or three parameters depending on in they are inheritable type of question. The parameter plugin is simply the reference to the MonocleGL plugin. The parameter configuration is the question level of a JSON object. The parameter thmDemo is JavaScript's reference to the demo and is used for inheritable questions to access the scene array.

To create a template question you must first create the question level JSON object for that question. You may then override the default scene in a demo with the template scene. As an example here's a simple ordering question JSON plus the code Javascript code necessary to override the third default question:

```
var orderJSON = {
	type:"ordering",
	text:"Q1) Order the events chronologically by dragging the items below and order
them by putting the first event and at the top and the last event at the bottom.",
	layout:"horizontal",
	x:"20",
	y:"20",
	width:"440",
	height:"220",
	objects:
	[{
		background_color:"DFCF99",
		rounded_corners:"15",
		text:"Introduction of the Federal Government Policy on Tobacco
Sponsorship of National Sport Organizations",
		text_layout:"full",
		text_align:"left",
		text_color:"000000",
		text_size:"12",
		vertical_padding:"2",
		horizontal_padding:"10"
	},{
		background_color:"DFCF99",
		rounded_corners:"15",
		text:"Introduction of the most recent Canadian Policy Against Doping in
Sport",
		text_layout:"full",
		text_align:"left",
			text_color:"000000",
			text_size:"12",
			vertical_padding:"2",
			horizontal_padding:"10"
		},{
			background_color:"DFCF99",
			rounded_corners:"15",
			text:"Introduction of Sport Canada's Policy on Aboriginal Peoples' 
Participation in Sport",
			text_layout:"full",
			text_align:"left",
			text_color:"000000",
			text_size:"12",
			vertical_padding:"2",
			horizontal_padding:"10"
		},{
			background_color:"DFCF99",
			rounded_corners:"15",
			text:"Introduction of the Policy on Sport for Persons with a Disability",
			text_layout:"full",
			text_align:"left",
			text_color:"000000",
			text_size:"12",
			vertical_padding:"2",
			horizontal_padding:"10"
		},{
			background_color:"DFCF99",
			rounded_corners:"15",
			text:"Introduction of Actively Engaged: A Policy on Sport for Women and 
Girls",
			text_layout:"full",
			text_align:"left",
			text_color:"000000",
			text_size:"12",
			vertical_padding:"2",
			horizontal_padding:"10"
		},{
			background_color:"DFCF99",
			rounded_corners:"15",
			text:"Hosting of the most recent Olympic and Paralympic Winter Games",
                    text_layout:"full",
                    text_align:"left",
                    text_color:"000000",
                    text_size:"12",
                    vertical_padding:"2",
                    horizontal_padding:"10"
          }]
};

thmDemo.getScene(2) = new THM_OrderingQuestion (plugin, orderJSON);
```