

Web Survey Creator

Introduction to Scripting



© Dipolar Pty Limited. All rights reserved.

Table of Contents

INTRODUCTION TO SCRIPTING	5
WHAT IS SCRIPTING?	6
<i>To Script or not to Script?</i>	<i>6</i>
AN INTRODUCTION TO JAVASCRIPT.....	7
<i>JavaScript is the most popular</i>	<i>7</i>
<i>JavaScript works Everywhere.....</i>	<i>8</i>
<i>JavaScript can react to Events.....</i>	<i>8</i>
JAVASCRIPT BASICS	8
<i>The <script> tag.....</i>	<i>8</i>
<i>JavaScript is Case Sensitive.....</i>	<i>8</i>
<i>JavaScript Code Essentials.....</i>	<i>9</i>
ADDING SCRIPTING IN WSC	15
<i>WSC Scripting Objects.....</i>	<i>15</i>
<i>How to Use a Question in Scripting</i>	<i>16</i>
<i>How to add Scripting to a Survey.....</i>	<i>16</i>
SCRIPTING #101: DEALING WITH DATA	19
WORKING WITH DATA IN SCRIPTING.....	20
<i>Rule #1: Script for each Question Type</i>	<i>20</i>
<i>Rule #2: Data Location effects Script syntax</i>	<i>21</i>
<i>Getting Data for Questions on the Current Page.....</i>	<i>21</i>
<i>Getting Data for Questions on a Previous Page</i>	<i>21</i>
READING AND WRITING DATA – THE BASICS.....	22
<i>Writing Data to a Text Question.....</i>	<i>22</i>
<i>Reading Data from a Text Question.....</i>	<i>24</i>
DATA SCRIPTING FOR OTHER QUESTION TYPES.....	25
<i>Choice Questions.....</i>	<i>26</i>
<i>How do Multi-Select Choice Questions Work?</i>	<i>28</i>
<i>Numeric Questions.....</i>	<i>29</i>
<i>Matrix Questions.....</i>	<i>29</i>
SCRIPTING #101: VALIDATION OF RESPONSES	31
WHAT DOES VALIDATION DO?	32
<i>Validation Example.....</i>	<i>32</i>
<i>Validations Available without Scripting</i>	<i>33</i>
VALIDATION USING SCRIPTING.....	34
<i>Why is scripted validation needed?.....</i>	<i>34</i>
<i>How Does Scripted Validation Work?</i>	<i>34</i>
<i>Scripted Validation Logic</i>	<i>34</i>
<i>What the Respondent Sees.....</i>	<i>35</i>
SCRIPTED VALIDATION EXAMPLE.....	35
<i>Preparing for our Validation Script.....</i>	<i>36</i>
<i>Writing the Script: Step-by-step</i>	<i>37</i>
<i>Putting it all together: The Final Script.....</i>	<i>39</i>

SCRIPTING #101: TWEAKING THE INTERFACE.....	41
OVERVIEW OF INTERFACE “TWEAKING”	42
CREATING CONTENT USING SCRIPTING.....	42
<i>Using a Content Container</i>	42
<i>Modifying Existing Content</i>	43
DEALING WITH UI EVENTS	45
<i>What Events can be Hooked into?</i>	45
<i>How can Events be Used?</i>	46
UI EVENT EXAMPLE	49
<i>Drop-down List with “Other”</i>	49
SCRIPTING #101: ORDERING OF PAGES & CHOICES.....	51
PAGE ORDERING IN THE DESIGNER.....	52
<i>Basic Page Order</i>	52
<i>Randomization of Pages</i>	52
<i>Another way to randomize - A/B Testing</i>	53
<i>Page Ordering through Scripting</i>	54
<i>Our Example: Ordering Pages based on Gender</i>	56
<i>Page Ordering in a Nutshell</i>	59
ORDERING CHOICES.....	60
<i>Standard Ordering of Choices</i>	60
<i>Setting up for Scripted Choice Ordering</i>	60
EXAMPLE: SCRIPTED CHOICE ORDERING	61
<i>Choice Questions</i>	61
<i>Single Range Matrix</i>	65
<i>Dual Range Matrix</i>	68
SCRIPTING REFERENCE.....	69
SCRIPTING OBJECTS.....	70
<i>args</i>	70
FUNCTION REFERENCE	74
ADDITIONAL OBJECTS	88
<i>SurveyQuestion</i>	88
<i>SurveyChoice</i>	90
<i>SurveyChoiceTag</i>	90
<i>SurveyRow</i>	91
<i>SurveyHierarchicalListItem</i>	91
<i>SurveyRowTag</i>	91
<i>SurveyQuota</i>	91
<i>SurveyDistribution</i>	92

Introduction to Scripting

The most advanced Web Survey Creation tools available today can provide an amazing amount of functionality through a simple, non-technical interface.

There are times, however, when very specific “one-off” functionality is needed in a survey. This is when scripting is needed.



What is Scripting?

For the purposes of this manual, we will only be considering scripting from the point of view of creating Web Surveys. All scripting in Web Survey Creator is created in **JavaScript**.

Scripting is simply a method of describing survey logic, or manipulating the survey interface, using an english-like “scripting language”.

When you use a script, you “explain” to the software how something needs to be performed in a powerful language, rather than using a pre-defined interface (which is limited to only allowing you to do whatever the interface was originally designed for).

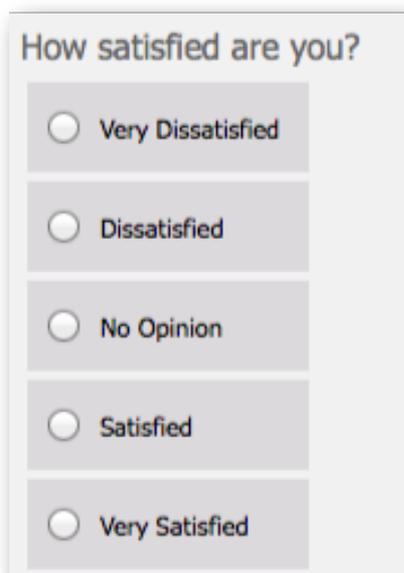
To Script or not to Script?

Scripting is great when there is no other way to achieve something. If there is another way, however, scripting adds complexity that would be good to avoid.

Scripting should only be used for functionality that is not possible through the standard survey design interface.

The good	The bad
Scripting is much more powerful than using a standard interface	You need to build the logic of the script yourself
Complex problems can be solved through scripting	You must understand the functions available and how to use them. You are responsible for ensuring the script is bug-free
All of the standard capabilities of JavaScript are available	There is a lot to learn to be able to benefit from all JavaScript has to offer

Let’s consider a simple example. I have a choice question as follows:



How satisfied are you?

Very Dissatisfied

Dissatisfied

No Opinion

Satisfied

Very Satisfied

If I want this question to default to the answer “No Opinion”, I can select this answer in script as follows:

```

var question = wscScripting.getQuestionByDataPipingCode('HOWSATISFIED');

if (question) {

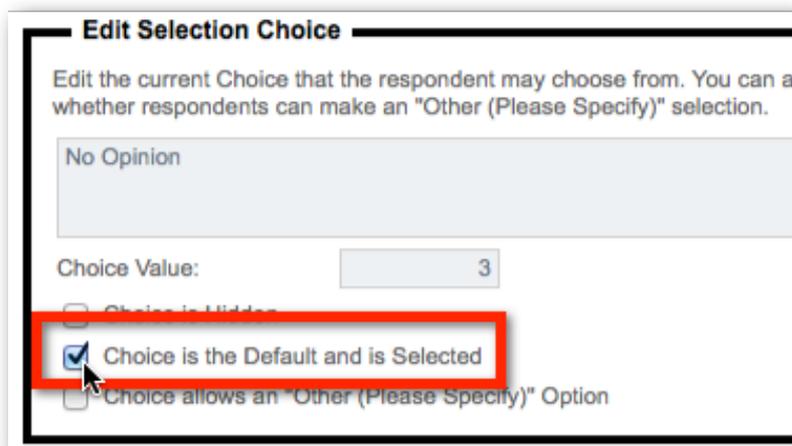
    var choice = wscScripting.getChoiceByValue(question, 3);

    if (choice) {

        var isSelected = wscScripting.selectChoice(question, choice);    } }

```

While this will work, it's a pretty complex way to set the value of the question. The easy way to do this is simply to make the "No Opinion" value a default value in the question:



This is an example of a simple rule when it comes to scripting:

**Just because you can do something in scripting doesn't necessarily mean you should.
Only use scripting to create functionality that is not available
to you in a non-scripted way.**

An Introduction to JavaScript

JavaScript is designed to add interactivity to Web pages. It is a lightweight programming language (also known as a "scripting language") that is embedded into Web pages.

Web Survey Creator was designed to use JavaScript for scripting for a number of reasons.

JavaScript is the most popular

JavaScript is the most popular scripting language on the Internet. This means that there are a large number of people who already know how to use it - if you don't, chances are you will easily be able to find someone who does.

This popularity is due in part to how easy it is to use JavaScript, so even if you don't know how to write in JavaScript yet, it won't take long to learn.

JavaScript works Everywhere

JavaScript works in all major browsers - for both PCs and mobile devices, including:

- Internet Explorer
- Firefox
- Chrome
- Safari
- Opera

This is important, because it means the scripts you write for your surveys will work everywhere.

JavaScript can react to Events

When adding a script to a survey, it is important to be able to control when a script is performed. For example, you may want to run a script as soon as a survey page is loaded.

JavaScript can react to events, so this sort of control is easy. Web Survey Creator allows you to choose when a script is run, including:

- When a Page is Loaded
- When Survey Quota Data is Loaded
- Testing visibility of Next or Submit buttons
- Before a Survey Page is Validated
- When Next or Submit buttons are pressed
- When Previous buttons are pressed

In addition to this, JavaScript can “hook in” to events like a radio button being pressed, or text being typed into a text field. This makes it possible to provide a high level of interactivity, as we will see later in this book.

JavaScript Basics

Before venturing into the world of Web Surveys, it is important to understand the fundamental concepts when using JavaScript.

The `<script>` tag

JavaScript is always contained within a `<script>` tag. Below is an example with a single line of JavaScript.

```
<script type="text/javascript">  
document.getElementById("demo").innerHTML=Date();  
</script>
```

Web Survey Creator includes the `<script>` tag automatically whenever you use script, so you will never need to put this tag in your own scripts.

JavaScript is Case Sensitive

Unlike some scripting languages, and HTML itself, JavaScript is case sensitive.

If you have written a piece of JavaScript and it looks right but is not working, the first thing to check is that you don't have the case wrong for any of the functions or variables.

Let's consider a simple piece of JavaScript that writes "Hello World" to the browser:

```
document.write("Hello World");
```

If this had been entered as follows, it simply wouldn't work:

```
document.Write("Hello World");
```

The only difference here is the capitalization of "Write".

```
document Write("Hello World");
```

You can see how easy it would be to overlook this error. When using any JavaScript function - including the specialized functions that have been created for Web Survey Creator, always follow a simple rule...

Always use the correct case for everything in your scripts!

JavaScript Code Essentials

JavaScript code is a sequence of JavaScript statements. They are executed in the order that they are written. It is best practice to end each statement with a semi-colon.

```
document.write("Hello World");  
document.write("This is written with JavaScript!");
```

Inserting Comments

If an explanation is needed, comments can be added to a script by starting a line with //.

```
// Let's write something to the browser...  
document.write("Hello World");  
document.write("This is written with JavaScript!");
```

If you want to add multiple lines of comments, you can start with /* and end with */.

```
/*  
Let's write something to the browser...  
This will add multiple lines to the browser  
*/  
document.write("Hello World");  
document.write("This is written with JavaScript!");
```

JavaScript Variables

JavaScript *variables* are used to hold values or expressions. A variable can have a short name, like *x*, or a more descriptive name, like *FavoriteColor*.

It is important to note that:

- Variable names are case sensitive (y and Y are two different variables)
- Variable names must begin with a letter, the \$ character, or the underscore character
- Variables are declared with the *var* keyword.

```
var x;  
var FavoriteColor;
```

After the declaration of a variable they are empty (they have no values yet). You can, however, assign values to the variables when you declare them:

```
var x = 100;  
var FavoriteColor = "Red";
```

As we will see later when we start building scripts, variables can be manipulated in various ways. For example, you can do arithmetic operations with variables:

```
var y = x - 100;  
var z = y + 100;
```

JavaScript Arithmetic Operators

Operator	Description	Example	Result
+	Addition	x=y+2	x=7 y=5
-	Subtraction	x=y-2	x=3 y=2
*	Multiplication	x=y*2	x=10 y=5
/	Division	x=y/2	x=2.5 y=5
%	Modulus (remainder)	x=y%2	x=1 y=5

Using the + Operator with Strings

When dealing with strings, the + operator can be used to join two or more strings. For example:

```
txt1="The weather looks";  
txt2="pretty good today";  
txt3=txt1+" "+txt2;
```

The result of this script is that the variable `txt3` will contain the text “The weather looks pretty good today”.

JavaScript Comparison Operators

Let’s assume we have a variable `x=5`.

Operator	Description	Example
<code>==</code>	is equal to	<code>x==5</code> is true
<code>===</code>	is exactly equal to (value and type)	<code>x===5</code> is true <code>x===”5”</code> is false
<code>!=</code>	is not equal	<code>x!=8</code> is true
<code>></code>	is greater than	<code>x>8</code> is false
<code><</code>	is less than	<code>x<8</code> is true
<code>>=</code>	is greater than or equal to	<code>x>=5</code> is true
<code><=</code>	is less than or equal to	<code>x<=4</code> is false

Comparison operators can be used in conditional statements to compare values and take action depending on the result.

We will see this in action when we start writing some scripts.

JavaScript Logical Operators

Let’s assume we have a variable `x=6` and `y=3`.

Operator	Description	Example
<code>&&</code>	and	<code>(x < 10 && y > 1)</code> is true
<code> </code>	or	<code>(x==5 y==5)</code> is false
<code>!</code>	not	<code>!(x==y)</code> is true

The JavaScript Conditional Operator

JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.

```
variablename=(condition)?value1:value2
```

For example:

```
var visitor="PRES";
var greeting=(visitor=="PRES")?"Dear President ":"Dear ";
document.write(greeting);
```

JavaScript Conditional Statements

Conditional statements are used to do different things in your script when different rules are met.

In JavaScript the following conditional statements can be used:

if statement - use this statement to execute some code only if a specified condition is true

if...else statement - use this statement to execute some code if the condition is true and another code if the condition is false

if...else if...else statement - use this statement to select one of many blocks of code to be executed

switch statement - use this statement to select one of many blocks of code to be executed

An example of a conditional statement is as follows:

```
var d = new Date()
var time = d.getHours()
if (time<10)
{
    document.write("<b>Good morning</b>");
}
else if (time>=10 && time<16)
{
    document.write("<b>Good day</b>");
}
else
{
    document.write("<b>Hello World!</b>");
}
```

The use of a *switch statement* can be a very efficient way to run the appropriate piece of code in your script.

This is how it works...

First we have a single expression *n* (most often a variable) that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use *break* to prevent the code from running into the next case automatically.

```
//You will receive a different greeting based
//on what day it is. Note that Sunday=0,
//Monday=1, Tuesday=2, etc.

var d=new Date();
var theDay=d.getDay();
switch (theDay)
{
case 5:
    document.write("Finally Friday");
    break;
case 6:
    document.write("Super Saturday");
    break;
case 0:
    document.write("Sleepy Sunday");
    break;
default:
    document.write("I'm looking forward to this weekend!");
}
```

JavaScript Loops

Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal lines in a script we can use loops to perform a task like this.

In JavaScript, there are two different kind of loops:

for - loops through a block of code a specified number of times

while - loops through a block of code while a specified condition is true

The format of a *for loop* is as follows:

```
for (variable=startvalue;variable<=endvalue;variable=variable+increment)
{
code to be executed
}
```

Here is an example of a *for loop*:

```
var i=0;
for (i=0;i<=5;i++)
{
document.write("The number is " + i);
document.write("<br />");
}
```

A *while loop* has a simpler structure. The format of a *while loop* is as follows:

```
while (variable<=endvalue)
{
code to be executed
}
```

Here is an example of a *while loop*:

```
var i=0;
while (i<=5)
{
document.write("The number is " + i);
document.write("<br />");
i++;
}
```

A variation of the *while loop* is the *do...while loop*. This will execute the block of code once, and then repeat it for as long as the specified condition is true.

```
var i=0;
do
{
  document.write("The number is " + i);
  document.write("<br />");
  i++;
}
while (i<=5);
```

The *break* statement will break the loop and continue executing the code that follows after the loop (if any).

```
var i=0;
for (i=0;i<=10;i++)
{
  if (i==3)
  {
    break;
  }
  document.write("The number is " + i);
  document.write("<br />");
}
```

The *continue* statement will break the current loop and continue with the next value.

```
var i=0
for (i=0;i<=10;i++)
{
  if (i==3)
  {
    continue;
  }
  document.write("The number is " + i);
  document.write("<br />");
}
```

Adding Scripting in WSC

Web Survey Creator allows scripting to be added like any other survey content. If you know how to add a question, you know how to add scripting!

Scripting is an extremely powerful feature, and is designed to be used in the two highest versions of the software - the MR Premium and MR Ultimate editions.

MR Premium Solo Our Premium MR Plan	MR Ultimate Solo The Ultimate MR Plan
Single User	Single User
Unlimited Respondents	Unlimited Respondents
Unlimited Questions	Unlimited Questions
10,000 Responses/Month *	20,000 Responses/Month *
30,000 Responses/Quarter *	60,000 Responses/Quarter *
120,000 Responses/Year *	240,000 Responses/Year *

WSC Scripting Objects

Scripts written in WSC have access to two specialized objects. These objects allow you access to the questions that are exposed on the current page and additional help methods that can help you to perform various tasks.

1. args
2. wscScripting

args

args contains a single item isValid that can be used to set the status of an event. This is particularly relevant for confirming to the event engine that you wish to continue the current process. For example, you must set the value to true on Next or Previous Button events or those processes are halted and will not continue.

Property:	isValid
Return Value:	boolean - Is the current process Valid
Example:	<pre>var isOkay = true; if (isOkay) { // All my changes allow me to continue args.isValid = true; }</pre>

wscScripting

The wscScripting object provides access to all the custom methods that have been set up for use in your scripts. These methods will be discussed in detail throughout this book.

How to Use a Question in Scripting

Scripting in a survey inevitably will need access to the basic elements in any survey - the questions. Accessing a question from within a script requires that the question is set up correctly.

There is only one thing you need to do to a question to make it ready for scripting - give the question a **Question Access Code**.

Question Access Code (Optional) [Used for Data Piping, SPSS, etc]: FROMQU

There are a few simple rules when adding access codes:

1. Each code must be unique to a particular question
2. Codes can only be characters and numbers, with no spaces
3. Codes must be at least 2 characters in length
4. Codes are enforced as upper case

Accessing a question is a simple process once the access code is set. A call to *getQuestionByDataPipingCode* returns the question object in the script.

```
// Get the question to load the value from  
var oFromTextQ = wscScripting.getQuestionByDataPipingCode('FROMQU');
```

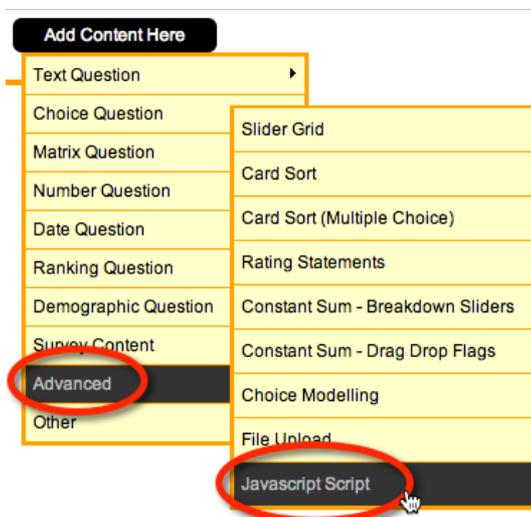
How to add Scripting to a Survey

Adding a Scripting Question in Web Survey Creator is similar to adding any other content in the system. The steps are as follows:

1. Press the **Add Content Here** button in the Survey Designer to add new content.



2. Choose **JavaScript Script Question** as the type of content you wish to add from the **Add Content Here** button menu.

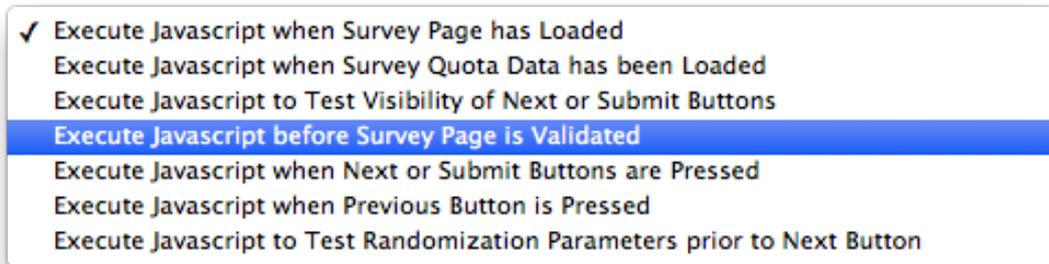


The content to be set consists of four parts:

1. **A description for the script** (straight text)
2. **Whether to show a container for the question.** You should show a container when you want to script something that shows on the survey page (as opposed to just scripting “behind the scenes” logic)



3. **A choice for when the script will be executed.** We will see in later chapters the reasons for choosing different execution times.



4. **The content of the script itself.** As previously mentioned, this is standard JavaScript without the `<script>` tag (or function references) - it is effectively the “guts” of the script we want to use. WSC will encapsulate it correctly on the page.

```
JavaScript to be Executed
Enter the Javascript to be executed. This script will be executed at point selected.
1 // Get the EMAIL question using it's data code
2 var oQuestion1 = wscScripting.getQuestionByDataPipingCode('EMAIL');
3 // Check if it is valid
4
5 // Mark it as initially false
6 args.isValid = false;
7
8 if (oQuestion1) {
```

That's it! Once added, a script will be executed by WSC at the time you have chosen.

The hard part is ensuring you know how to write the script. That is the subject of the rest of this book. Happy scripting!

Scripting #101:

Dealing with Data

One of the most common uses of scripting is to perform some logic based upon responses entered into survey questions. This requires data to be “read” by a script.

It is also common to modify the data entered in a survey through scripting. This requires data to be “written” by a script. This chapter discusses how to “read” and “write” survey data through scripting.



Working with Data in Scripting

Loading and saving survey data with scripting is relatively simple. There are a couple of basic rules you must understand, however, before diving into scripting.

Rule #1: Script for each Question Type

The type of question determines how loading & saving must be implemented in script.

For example, a *text question* simply stores a piece of text, therefore the script to load and save values looks as follows:

```
// Get the question to load the value from
var oFromTextQ = wscScripting.getQuestionByDataPipingCode('FROMQU');
// Get the value from the question
var oGetValue = wscScripting.getValue(oFromTextQ);

// Get the question to write the value to
var oToTextQ = wscScripting.getQuestionByDataPipingCode('TOQU');
// Save the value to the question
wscScripting.setValue(oToTextQ, oGetValue);
```

The script above is very straightforward, since a text question has a single text value. Let's consider a *multi-selection choice question*, however. The answer to this type of question can be multiple choices.

In order to get the value from one multiple choice question and save it to a new multiple choice question, we would need to do more work than we had to do for the text question. We would need to loop through all the possible choices and work out which choices were selected. These choices would then need to be selected in the new question.

```
// Get the question to load the value from
var oFromTextQ = wscScripting.getQuestionByDataPipingCode('FROMQU');
// Get the value from the question
var oGetValue = wscScripting.getValue(oFromTextQ);

// Get the question to write the value to
var oToTextQ = wscScripting.getQuestionByDataPipingCode('TOQU');
// Save the value to the question
wscScripting.setValue(oToTextQ, oGetValue);

// Get the question to load the value from
var oFromMultiChoice = wscScripting.getQuestionByDataPipingCode('FROMQU');
// Get the question to write the value to
var oToMultiChoice = wscScripting.getQuestionByDataPipingCode('TOQU');

// Loop through the values in the question
for (i=1; i<=oToMultiChoice.choices.length; i++)
{
    // See if the value is selected in the from question
    var oSelected = wscScripting.isChoiceSelectedByValue(oFromMultiChoice, i);
    if (oSelected)
    {
        // The value was selected - make it selected in the to question
        wscScripting.selectChoiceByValue(oToMultiChoice, i);
    }
}
}
```

As we can see, it's a bit more complex for the more complex question types. The principles are the same however - you just have to methodically work through the correct way to deal with the data.

Rule #2: Data Location effects Script syntax

Dealing with question data through scripting has to work within the technical boundaries set by how JavaScript operates.

So what does this mean exactly?

JavaScript scripting runs on the browser. While technically speaking, JavaScript could access data from anywhere, speed and security considerations dictate that scripting can only get to data that is available on the current page.

Getting Data for Questions on the Current Page

If we are restricted to getting data from the current page, questions on the current page should be easy to get to, right?

Absolutely!

In fact, the previous examples all access data from questions on the current page. And regardless of the question type, getting the question is always the same:

```
// Get the question to load the value from
var oFromTextQ = wscScripting.getQuestionByDataPipingCode('FROMQU');
```

This is really easy. Unfortunately it's also relatively useless in real-world use, since you will almost always want to load data from a question that is not on the current page...

Getting Data for Questions on a Previous Page

If you are wishing to read up the data for a question, more often than not this question will appear on a *prior page of the survey*. The data for this question will not be available on the current page, because the question is not on the current page.

To access data for a question on a previous page, the script needs to explicitly indicate that the data will need to be loaded. If we look at our previous example, the script would need to change as follows if the question was on a prior page:

```
// Get the question to load the value from
var oFromTextQ = wscScripting.getQuestionByDataPipingCode('@FROMQU#DATA#@');
```

The code for the question takes the format of a data piping code if the question is on a different page. This tells the system that it has to load the data for this question onto this page because the script is going to need it.

Apart from the syntax for the code to access the question, everything else works exactly the same in the script.

Reading and Writing Data – the Basics

So far we have looked at script fragments to explain some basic scripting functionality. Now let's look in detail at *full scripts* for read and write data to and from questions with scripting.

For the purposes of these examples, we will use the simplest question type for scripting - text questions.

Writing Data to a Text Question

Let's look at all the steps needed to write data to a question in a survey - starting from the beginning.

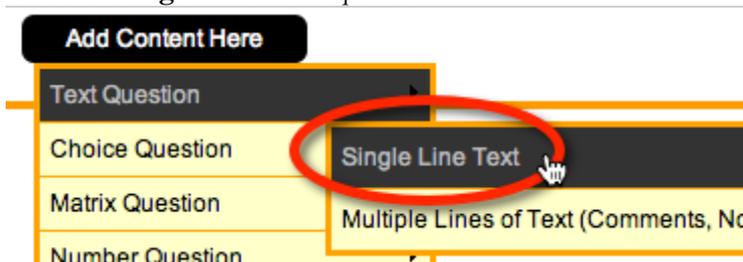
Creating a Text Question

The creation of questions is covered earlier in this book, but the key steps are:

1. Press the **Add Content Here** button.



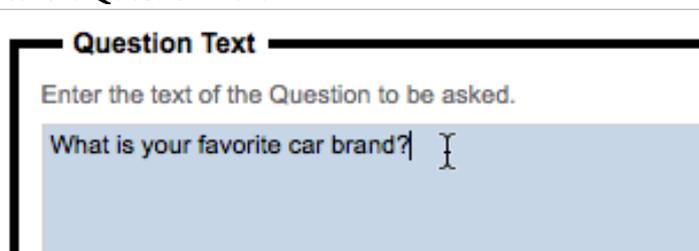
2. Choose a **Single Line Text** question.



3. Enter a **Question Access Code** for the question. Note that this code is used in scripting - without it, this question can not be accessed in a script.



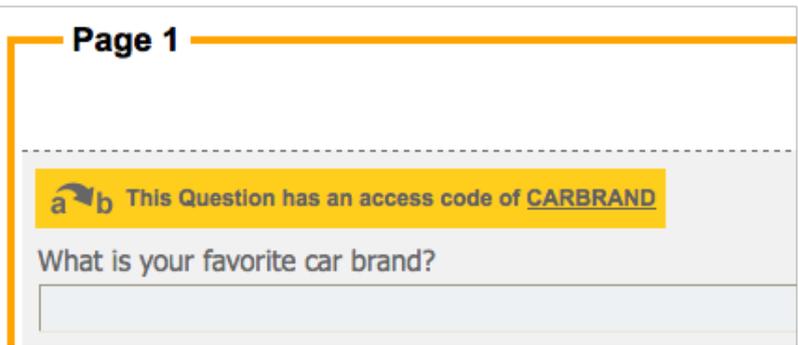
4. Enter the **Question Text**.



5. Press the *Save Content* button.



The question will be shown in the designer. The unique access code is displayed prominently above the question.



Writing Data to a Text Question

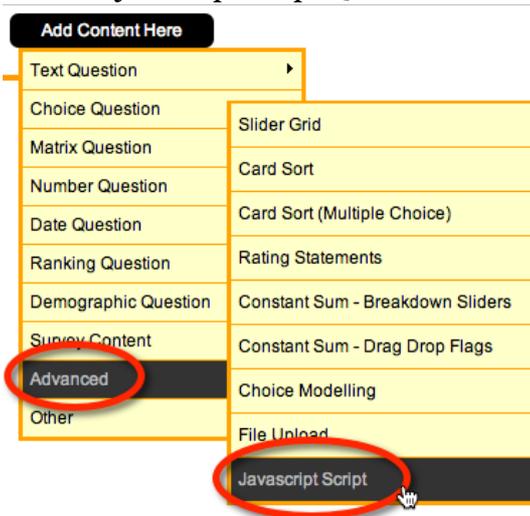
To write data to our text question, we need to add some scripting. Let's assume that we want to set the text question to "I don't know" when the page loads.

The steps to set up this script are as follows:

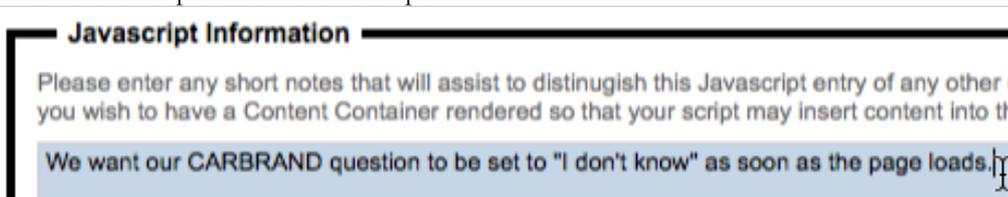
1. Press the **Add Content Here** button.



2. Choose a **Javascript Script Question**.



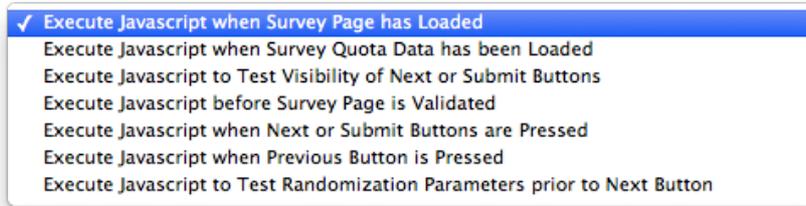
3. Enter a brief explanation of the script.



4. We don't want to create any custom interface on the survey page using the script, so we don't need a content container.



- Choose to execute the script when the page is loaded.



- Enter the script.

```

Javascript to be Executed
Enter the Javascript to be executed. This script will be executed at point selected.
1 // Get the question
2 var oTextQuestion = wscScripting.getQuestionByDataPipingCode('CARBRAND');
3
4 if (oTextQuestion ) // The question was loaded successfully
5 {
6
7 // Set the question value. Note the use of \' in the text, since
8 // using \' in a string is not directly allowed
9
10 wscScripting.setValue(oTextQuestion, 'I don\'t know');
11
12 }
13
14 // Always return true unless you want to indicate a failure
15 // in the script
16 args.isValid = true;

```

- Press the **Save Content** button.



Script Elements

The table below highlights fragments of the script that are important to learn and understand.

We want to...	Key script Fragment...
Load the question to use	wscScripting.getQuestionByDataPipingCode
Write to the question	wscScripting.setValue

Reading Data from a Text Question

Let's now consider how we can read data from a text question. For the purposes of our example, we will have a two-page survey, with each page containing a single text question:



We have already created the first question. The creation of the second question is exactly the same, except we give it a unique code of CARBRAND2.

What we now want to do is fill the second question with whatever the text is in the first question. This will mean that we have to *read* from CARBRAND and *write* to CARBRAND2.

The script to do this looks as follows:

```
// Get the question we want to load the data from
// It's on a different page, so note the syntax of the code
var oFromTextQuestion = wscScripting.getQuestionByDataPipingCode( '@CARBRAND#DATA#@' );

if ( oFromTextQuestion ) // The question was loaded successfully
{

    // Get the value from the question
    var oGetValue = wscScripting.getValue(oFromTextQuestion);

    // Get the question we want to save the data to
    // It's on a this page, so note the syntax of the code
    var oToTextQuestion = wscScripting.getQuestionByDataPipingCode( 'CARBRAND2' );

    if ( oToTextQuestion ) // The question was loaded successfully
    {

        wscScripting.setValue(oToTextQuestion, oGetValue);

    }

}

// Always return true unless you want to indicate a failure
// in the script
args.isValid = true;
```

Script Elements

The table below highlights fragments of the script that are important to learn and understand.

We want to...	Key script Fragment...
Load a question from previous page	Must use [@CARBRAND#DATA#@] syntax
Read the value of a question	wscScripting.getValue

Data Scripting for Other Question Types

Scripting is all about taking what you already know, and making a couple of variations, or adding a small piece of additional functionality. This is why it is important to have a strong knowledge of the basics, and build from there.

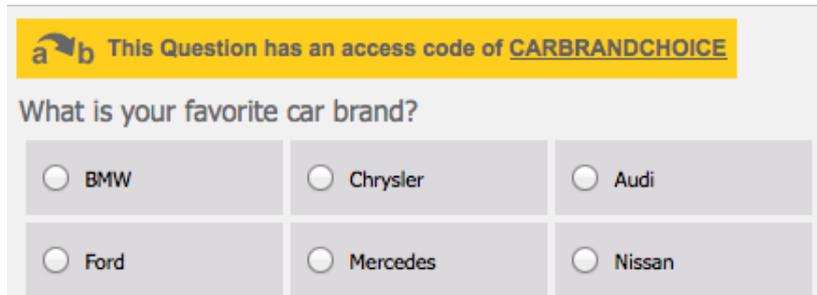
We now know how to save data to and get data from a text question. All other questions will be a variation on this knowledge. Let's look at the other key question types you will want to access through scripting.

Choice Questions

Choice Questions are a little trickier than text questions. Rather than a flat piece of text, the “data” that must be saved or loaded in a script is a “choice” (or a number of choices if the question allows multiple selections). So this begs the question...

How do we manage choices in scripting?

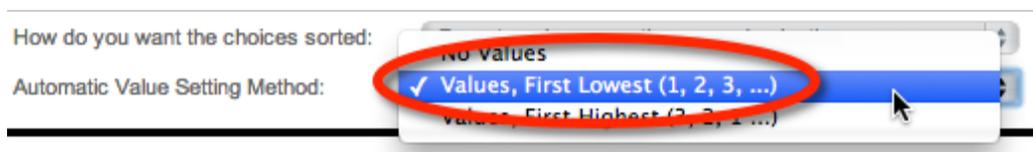
To answer this question, let’s consider an example.



This is a choice question with a unique code of *CARBRANDCHOICE*. We already know how to get the question in script (just like we did with the text questions!) What we now need to understand is how to set the value of this type of question in script.

Attaching Values to Choices

There is one other thing that should be done when setting a choice question up to make scripting easier - allocating each choice a numeric value. This can be achieved by choosing to automatically apply values to the choices entered when the question is first added.



By doing this, our choices in this example question would be as follows:

Value	Description
1	BMW
2	Ford
3	Chrysler
4	Mercedes
5	Audi
6	Nissan

Choices with values can be managed directly in script by referring to those values. It is therefore best practice to do two things when creating any question that uses choices that you wish to manipulate in scripting:

1. Give each choice a numeric value
2. Make sure each of these values is unique

Writing Data to a Choice Question

Let's assume we want to set our sample question to "Audi". The script to achieve this would be as follows:

```
// Get the question
var oChoiceQuestion = wscScripting.getQuestionByDataPipingCode('CARBRANDCHOICE');

if (oChoiceQuestion ) // The question was loaded successfully
{

    // Audi has the value 5. Let's set the answer for the question to 5.
    var isSelected = wscScripting.selectChoiceByValue(oChoiceQuestion, 5);

}

// Always return true unless you want to indicate a failure
// in the script
args.isValid = true;
```

Script Elements

The table below highlights fragments of the script that are important to learn and understand.

We want to...	Key script Fragment...
Choose the choice to set the question to	wscScripting.selectChoiceByValue

Reading Data from a Choice Question

Reading data for a choice question is simply a matter of looping through each of the choices, and seeing if they are actually selected.

```
// Get the source question (we want to read from)
var oChoiceQuestion = wscScripting.getQuestionByDataPipingCode('[@CARBRANDCHOICE#DATA#&]');

if (oChoiceQuestion ) // The question was loaded successfully
{
    // Get the target question (we want to write to)
    var oChoiceQuestion2 = wscScripting.getQuestionByDataPipingCode('CARBRANDCHOICE2');

    if (oChoiceQuestion2 ) // The question was loaded successfully
    {
        for (i=1; i<=oChoiceQuestion.choices.length; i++) // Loop through the choice values
        {
            var oSelected = wscScripting.isChoiceSelectedByValue(oChoiceQuestion, i);

            if (oSelected) // What the item we are looking at selected?
            {
                // Yes it was - select the same choice in the target question
                wscScripting.selectChoiceByValue(oChoiceQuestion2, i);
            }
        }
    }
}

// Always return true unless you want to indicate a failure
// in the script
args.isValid = true;
```

Script Elements

The table below highlights fragments of the script that are important to learn and understand.

We want to...	Key script Fragment...
Create a loop to go through the choices	for (i=1; i<=oChoiceQuestion.choices.length; i++)
Test if a choice was selected	wscScripting.isChoiceSelectedByValue

How do Multi-Select Choice Questions Work?

Reading and writing to multi-select choice questions work in a similar way to single select questions, except that you have to manage the fact that multiple items can be selected.

In single select question, selecting a particular value deselects other values automatically because by definition there can be only one value. In multi-select questions, you need to manage the de-selection of values yourself.

In the previous example, if we were dealing with multi-select questions, we would have to modify the code as follows:

Instead of:

```
var oSelected = wscScripting.isChoiceSelectedByValue(oChoiceQuestion, i);

if (oSelected) // What the item we are looking at selected?
{
    // Yes it was - select the same choice in the target question
    wscScripting.selectChoiceByValue(oChoiceQuestion2, i);
}
```

We would need to deal with unselected values too:

```
var oSelected = wscScripting.isChoiceSelectedByValue(oChoiceQuestion, i);

if (oSelected) // What the item we are looking at selected?
{
    // Yes it was - select the same choice in the target question
    wscScripting.selectChoiceByValue(oChoiceQuestion2, i);
}
else
{
    // No it wasn't - deselect the same choice in the target question
    wscScripting.deselectChoiceByValue(oChoiceQuestion2, i);
}
```

Script Elements

The table below highlights a fragment of the script that is important to learn and understand.

We want to...	Key script Fragment...
Deselect a choice that was not chosen	wscScripting.deselectChoiceByValue

Numeric Questions

When you are dealing with numeric questions, they are handled in exactly the same way as text questions. As you will note from the example below, you even put the numbers to place in the question in quotes ("), just like text values.

```
// Get the question
var oNumberQuestion = wscScripting.getQuestionByDataPipingCode('CARPRICE');

if (oNumberQuestion ) // The question was loaded successfully
{

    // Set the starting number for this question to $25000
    wscScripting.setValue(oNumberQuestion, '25000');

}

// Always return true unless you want to indicate a failure
// in the script
args.isValid = true;
```

There are a number of question types that are effectively numeric questions, and therefore behave in the same way as shown above. These include star rating and slider questions.

Matrix Questions

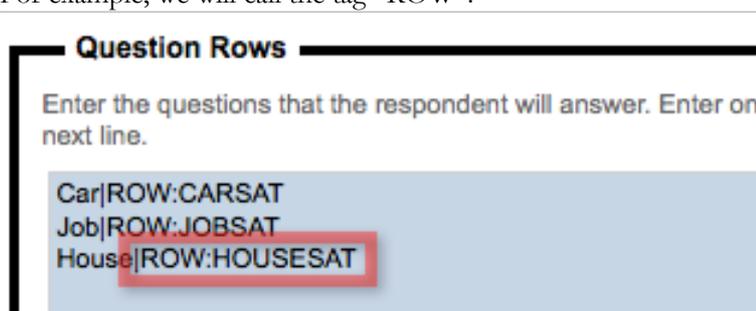
Matrix Questions are one of the more complex question types, since each matrix row is a question by itself. We know how to get to a question in a script - using the *Unique Access Code*. The question is, how do we get to an individual row?

Getting to a Matrix Row

Each row of a matrix can be thought of as its own choice question. If we could get to a row, we could manipulate it in a similar way to a standard choice question.

Matrix rows do not have their own Access Codes, so there needs to be another way to uniquely identify a row. To identify and use a row, the key steps are as follows:

1. When creating rows in a matrix, each row you want to access needs to be given a tag with a value. For example, we will call the tag "ROW":



The screenshot shows a form titled "Question Rows" with the instruction "Enter the questions that the respondent will answer. Enter on next line." Below this, there is a list of items: "Car|ROW:CARSAT", "Job|ROW:JOBSAT", and "House|ROW:HOUSESAT". The "House|ROW:HOUSESAT" entry is highlighted with a red rectangular box.

2. Get to the row in your script

```
var oMatrix = wscScripting.getQuestionByDataPipingCode('MYMATRIX');  
  
if (oMatrix)  
{  
    var oRow = wscScripting.getRowByTagValue(oMatrix, 'ROW', 'JOBSAT');  
}
```

Write to, or read from, a Matrix Row

Below is a code snippet showing how you can confirm if a value has been selected in a grid, and how to set a value for a row in a grid.

```
//Get the Job Satisfaction Row  
var oRow = wscScripting.getRowByTagValue(oMatrix, 'ROW', 'JOBSAT');  
// Are values 1 or 2 selected?  
var isSelected = wscScripting.isAnyMatrixChoiceSelectedByValue(oMatrix, new Array(1,2), oRow)  
  
// Is Value 3 selected  
var isSelected2 = wscScripting.isMatrixChoiceSelectedByValue(oMatrix, 3, oRow )  
  
// Select Value 4 in the Row  
var isSelected3 = wscScripting.selectMatrixChoiceByValue(oMatrix, 4, oRow )
```

Script Elements

The table below highlights fragments of the script that are important to learn and understand.

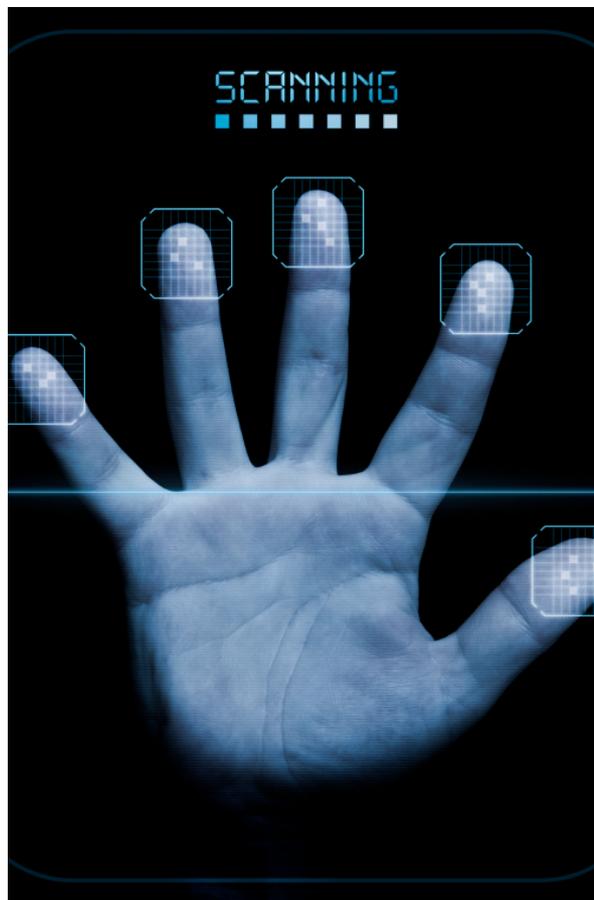
We want to...	Key script Fragment...
Get the row to read from/write to	wscScripting.getRowByTagValue
Check if one of a number of values is selected	wscScripting.isAnyMatrixChoiceSelectedByValue
Check if a single value is selected	wscScripting.isMatrixChoiceSelectedByValue
Set the value for a matrix row	wscScripting.selectMatrixChoiceByValue

Scripting #101:

Validation of Responses

Web Survey Creator provides all the standard built-in validation capabilities that you would expect from a high-end MR Survey Tool. There's often a "one-off" validation, however, that needs to be scripted because it is so specific to the survey at hand.

In this chapter we look at how you can set up custom validation within your surveys using scripting.



What does Validation do?

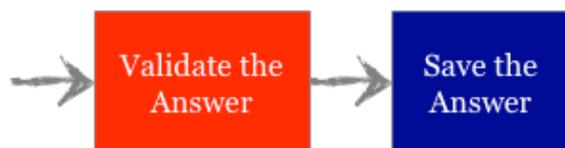
When considering the lifecycle of a typical question in a typical survey, it looks something like this:



This is nice and simple, but doesn't deal with a key question:

What if the answer entered is invalid?

This is where validation comes in - we want to “validate” that a response is correct before saving it. The validation process therefore occurs just before we save the response:



Validation Example

One of the simplest examples of validation is *Mandatory checking*. This involves checking that an answer has actually been entered.

For example, a survey may ask for the name of the respondent. This could be used to identify who created a particular response. It is therefore important that the name is actually entered. Placing a mandatory validation on the name question will ensure that a respondent can not continue until a name is entered.

In Web Survey Creator, validations are run as soon as a respondent moves off a survey page (by pressing the *Next button* or *Submit button*).

The screenshot shows a survey form with the title "Please enter your name". Below the title are three input fields: "Title", "First Name", and "Last Name". At the bottom of the form, there are two buttons: "Previous" on the left and "Submit" on the right. The "Submit" button is circled in red, indicating that it is the button that triggers the validation process.

If a validation fails, the survey does not advance, and a warning is provided for the question to explain why the validation failed.

This question is mandatory and you must complete an answer.

Please enter your name

Validations Available without Scripting

Web Survey Creator has a series of standard validations for the various types of questions available in the software. Checking a box indicating that the validation should be tested, and setting the related details for the validation can set up these validations.

Number Validation

Number Range must be validated
 You can specify the minimum and maximum value that a number has to be.

Number Range: Between And

Displayed Error Message when number is outside of range: This answer may only contain a numeric value between {NumberMin} and {NumberMax}

Validations all feature a validation message that is shown if the validation fails. Web Survey Creator has standard messages for all validations in more than 10 of the most common languages. These messages can be changed if required as part of the validation setup.

The standard validations available for different question types are shown below.

TEXT QUESTIONS		
	Mandatory	
Single & Multi-line	Format	Email address, text only, numeric only, URL, phone number, zip code, social security number
	Length	Minimum, maximum

CHOICE QUESTIONS		
Single Selection	Mandatory	
Multi Selection	Mandatory	
	No. of Selections	Minimum, maximum

NUMERIC QUESTIONS		
	Mandatory	
Numeric	Format	Integers, decimals, US currency, Euro currency, integer percentage, decimal percentage
	Range	Minimum, maximum

Most other question types simply have mandatory validations.

Validation using Scripting

Why is scripted validation needed?

The standard validations are often sufficient to ensure that the data entered into a survey is valid. For more advanced surveys, however, they do have a number of limitations:

- The “logic” they apply is quite rudimentary
- They don’t allow validations between questions

Scripted validation is thus needed to take validations past these limitations.

How Does Scripted Validation Work?

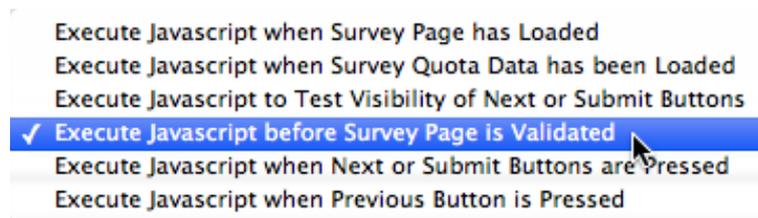
All validations have two processes to complete:

1. Perform the *validation logic*
2. If the logic fails:
 - a. Show a warning message; and
 - b. Halt the progress of the survey

A scripted validation changes the first step. It replaces the simplified logic that can be created through a couple of clicks with a much broader, more capable logic that is created by using a script.

Scripted Validation Logic

Scripts can be set up to run at different times. A validation script must be set up to run *before the Survey Page is Validated*.



This means that the script is run after the next or submit button is pressed, and before WSC moves to a new page (or completes the survey).



The contents of the validation script are completely up to you. All WSC cares about is:

1. Whether the validation is passed or failed.
2. If the validation failed, what validation error text should be shown?

If the validation is passed, the script must set:

```
args.isValid = true;
```

If the validation is failed, the script must set:

```
args.isValid = false;
```

Failure to validate will halt the survey on the current page. The script needs to tell WSC what validation text needs to be shown, and on which question:

```
wscScripting.setValidation(oQuestion, 'This is not valid!');
```

We will see how all this works together in the next section of this book.

What the Respondent Sees...

A respondent only sees the result of the second part of the validation process - the warning message, and the halted survey. Therefore, from the perspective of a survey respondent, a scripted validation will look exactly the same as a standard validation - it will appear in a box above the question that is being validated.



Scripted Validation Example

We want to survey people about air travel. This survey relates specifically to the last time they took a flight domestically in Australia. We are looking to achieve the following functionality in our survey:

Two of the key questions we need to ask are:

1. Which Australian destination did you fly FROM?
2. Which Australian destination did you fly TO?

We want to make sure that people enter valid data. Specifically, someone can't depart from and fly to the same destination.

The two questions look as follows:

A screenshot of a survey question titled "Which Australian destination did you fly FROM?". The question is presented in a light gray box. Below the title, there are seven radio button options arranged in three rows: Adelaide, Brisbane, Cairns in the first row; Canberra, Darwin, Melbourne in the second row; and Perth, Sydney in the third row.

Which Australian destination did you fly TO?

<input type="radio"/> Adelaide	<input type="radio"/> Canberra	<input type="radio"/> Perth
<input type="radio"/> Brisbane	<input type="radio"/> Darwin	<input type="radio"/> Sydney
<input type="radio"/> Cairns	<input type="radio"/> Melbourne	

Preparing for our Validation Script

Before we focus on the content of our script, we need to prepare for it.

Giving questions unique access codes

We will give our questions the access codes:

DEPARTURECITY
ARRIVALCITY

We add them to the questions and they are visible in the designer so we can be sure they have been added:

This screenshot shows two survey questions in a designer interface. The first question is "Which Australian destination did you fly **FROM**?" and its access code, "DEPARTURECITY", is circled in red. The second question is "Which Australian destination did you fly **TO**?" and its access code, "ARRIVALCITY", is also circled in red. Between the two questions, there are three buttons: "Add Content Here", "Group Content", and "S".

Making sure choices have values

The easiest way to access choices through scripting is to refer to each choice by its numeric value. We will therefore make sure that our choices have values.

This is done when the choice are originally added:

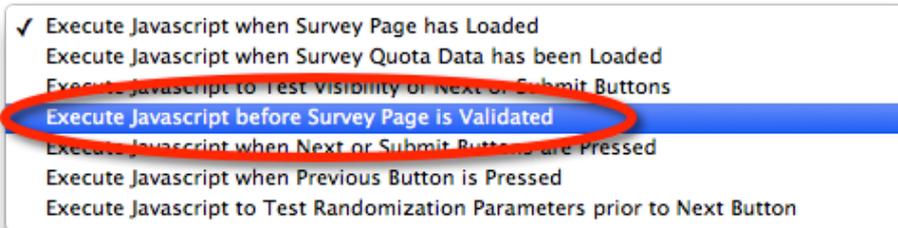
This screenshot shows the "Automatic Value Setting Method" dropdown menu in a survey designer. The menu is open, showing three options: "No values", "Values, First Lowest (1, 2, 3, ...)", and "Values, First Highest (3, 2, 1, ...)". The "Values, First Lowest (1, 2, 3, ...)" option is selected and circled in red.

Writing the Script: Step-by-step

Our validation script uses things we have learned in the previous chapter about loading question data, together with the validation methodology from the previous section in this chapter. Let's work through the script a step at a time.

Choose when the script will run

When we create our script, we must make sure that it runs at the appropriate time. Specifically, it needs to run the script *before the Survey Page is Validated*.



Get the Questions

Validations always require data to work with. Comparison of an answer to some set of rules clearly requires the loading of the questions by the script. In our example, we need to load two questions:

```
// Get our Questions
var oDep = wscScripting.getQuestionByDataPipingCode('DEPARTURECITY');
var oArr = wscScripting.getQuestionByDataPipingCode('ARRIVALCITY');
```

Test the Data

The testing of the question data is the key function of this script. We need to work out:

What is the best way to determine if the same choice has been made for both questions?

The wscScripting object allows us to test whether a certain value has been selected:

```
var oSelected = wscScripting.isChoiceSelectedByValue(
```

The easiest way to tell if two choice questions have the same choice selected is to loop through the choices one at a time. A standard loop would look as follows:

```
for (i=1; i<=oQuestion.choices.length; i++)
```

This loop will count from 1 up to the total number of choices for the question.

So, if we put all of the knowledge we have so far together, we could create a script to test the data as follows:

```

// Get our Questions
var oDep = wscScripting.getQuestionByDataPipingCode('DEPARTURECITY');
var oArr = wscScripting.getQuestionByDataPipingCode('ARRIVALCITY');

if (oDep && oArr)
{
    // Loop through the choices...
    for (i=1; i<=oDep.choices.length; i++)
    {
        // Was this choice selected as the departure city?
        var oDepSelected = wscScripting.isChoiceSelectedByValue(oDep, i);
        // Was this choice selected as the arrival city?
        var oArrSelected = wscScripting.isChoiceSelectedByValue(oArr, i);

        if ( oDepSelected && oArrSelected) // Same selection! Invalid!
        {
            // Need to throw a validation error
        }
        else
        {
            // No error
        }
    }
}

```

This is looking great - now we need to deal with the validation itself.

Stop the Survey

If the data is invalid, we need to stop on the current page of the survey rather than going to the next page, or submitting. This is achieved by ensuring that:

```
args.isValid = false;
```

This is quite different from most scripts we write. In fact, unless we want to indicate an error such as a validation failure, or scripts normally end in:

```
args.isValid = true;
```

Show the Validation Message

If the survey just stopped in its tracks with no warning, this would be very off-putting to a respondent. We need to ensure that the script will provide a visual warning for the validation so that the respondent knows what is going on.

This is achieved by attaching a validation message to one or more questions to indicate what the validation error is.

```
wscScripting.setValidation( oQuestion, 'Not Valid!');
```

Once a validation message has been attached to a question, it will remain attached to it until cleared. It is therefore important to ensure that any validation message that may have been previously added to a question is cleared if the question now passes validation.

```
wscScripting.clearValidation( oQuestion);
```

Putting it all together: The Final Script

We now have all the pieces to create the custom validation. Our final validation script would look as follows:

```
// Get our Questions
var oDep = wscScripting.getQuestionByDataPipingCode('DEPARTURECITY');
var oArr = wscScripting.getQuestionByDataPipingCode('ARRIVALCITY');
var bValid;

if (oDep && oArr)
{
    // Loop through the choices...
    for (i=1; i<oDep.choices.length; i++)
    {
        // Was this choice selected as the departure city?
        var oDepSelected = wscScripting.isChoiceSelectedByValue(oDep, i);
        // Was this choice selected as the arrival city?
        var oArrSelected = wscScripting.isChoiceSelectedByValue(oArr, i);

        if ( oDepSelected && oArrSelected) // Same selection! Invalid!
        {
            // Need to throw a validation error
            wscScripting.setValidation(oArr, 'Invalid location choice!');
            bValid = false;
            break; // No need to continue looping - error found
        }
        else
        {
            // Clear the validation text
            wscScripting.clearValidation(oArr);
            bValid = true;
        }
    }
}
args.isValid = bValid;
```

The interface that will be seen by a respondent when invalid data is entered will look as follows:

Which Australian destination did you fly **FROM**?

<input type="radio"/> Adelaide	<input type="radio"/> Canberra	<input type="radio"/> Perth
<input checked="" type="radio"/> Brisbane	<input type="radio"/> Darwin	<input type="radio"/> Sydney
<input type="radio"/> Cairns	<input type="radio"/> Melbourne	

Invalid location choice!

Which Australian destination did you fly **TO**?

<input type="radio"/> Adelaide	<input type="radio"/> Canberra	<input type="radio"/> Perth
<input checked="" type="radio"/> Brisbane	<input type="radio"/> Darwin	<input type="radio"/> Sydney
<input type="radio"/> Cairns	<input type="radio"/> Melbourne	

Previous Next

Scripting #101:

Tweaking the Interface

JavaScript is a client-side scripting language, which makes it perfectly placed to control and manipulate what is shown in the browser.

This chapter looks at some examples of what you can do to the survey interface through scripting.



Overview of Interface “Tweaking”

Interface “tweaking” is definitely an advanced topic, particularly when it comes to playing with existing interface elements in a survey. Compared to the other uses of scripting we have discussed so far, interface “tweaking” is more complex, less structured, and more directly tied to your level of knowledge of HTML, CSS and JavaScript.

The scripts demonstrated in this chapter are merely a couple of examples of what is possible.

The most common things you can do to the interface using scripting fall under the following headings:

1. Create new content (e.g. HTML Content on a page)
2. Modifying Existing Content (e.g. modifying the layout of existing questions)
3. Creating interactive interface elements (e.g. reacting to a button click)

We will consider examples of each of these in this chapter.

Creating Content Using Scripting

The simplest form of interface “tweaking” is the adding of custom content to a survey page. All the script requires in these situations is a container on the page for the script to “hook” on to.

Using a Content Container

In order to use a content container on a page, there are two steps:

1. Tick the “Show Content Container” checkbox on the script settings.



2. Access the container in the script. Note that the container for the current script is always accessed with the following code using {QuestionContainer}.

```
var oContainer = wscScripting.getElementById('{QuestionContainer}');
```

Once we have the container in script, we can do anything we like with it. For example, we could place some simple HTML in it..

```
var oContainer = wscScripting.getElementById('{QuestionContainer}');

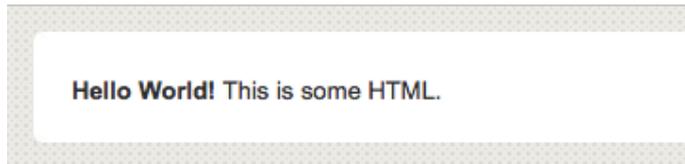
if (oContainer)
{
    // NOTE: when adding to innerHTML - put everything in a temp
    // variable first, as innerHTML will attempt
    // to complete missing tags for each line you add to it.
    var content = '';

    content += '<strong>Hello World!</strong> This is some HTML.';

    oContainer.innerHTML = content;
}

args.isValid = true;
```

This script will result in the following content being shown on the page:



Modifying Existing Content

For the purposes of this discussion, let's look at a simple example of modifying existing content. For this example, we will start with the following question:

There have been many changes within our company over the past two years to help streamline our operations and create a strong financial base for the future. Do you feel the changes that have been made have affected how you view our company in a negative way?

YES

NO

Modifying Question Layout

Questions shown in Web Survey Creator provide a number of layout options, including how wide to make the question, and whether to show values across the page. In this case we want to do something a little different - *we want to center the Yes and No options in the middle of the page.*

The choices in this question are in a table, so what we need to do in our script is “hook in” to that table and modify it so that it will be centered.

We have given the question a *unique code* of YESNO. The script (which will be run on page load) needed to get the table for this question, and make it centered, is as follows:

```

// Get our question object
var oVote = wscScripting.getQuestionByDataPipingCode('YESNO');

if (oVote) // If the question object is valid...
{
  // Get the table used for the question
  var oTable = wscScripting.getElementById(oVote.identity + '_table');

  if (oTable)
  {
    // Do something with the table
    // Change the left and right margins
    oTable.style.marginLeft = 'auto';
    oTable.style.marginRight = 'auto';

  }
}

args.isValid = true;

```

By using this script, we have changed the layout of the question so that it now looks as follows:

There have been many changes within our company over the past two years to help streamline our operations and create a strong financial base for the future. Do you feel the changes that have been made have affected how you view our company in a negative way?

YES

NO

Hiding the Previous Button

Web Survey Creator gives you the option of hiding or showing previous buttons on your survey pages. This is a global setting, though. What if I want to show previous buttons on all but one page?

Fortunately all aspects of the interface are accessible to the script. Let's write a script that hides the previous button on a single page.

Once we understand how to get access to the previous button in script, this becomes a very simple script to write:

```
// Get the previous button
var oButton = wscScripting.getElementById('previousbutton');

if (oButton) // If the button object is valid...
{
    // Change the button style so the button is not shown
    oButton.style.display = 'none';
}

args.isValid = true;
```

The page now looks as follows (no previous button is shown):

There have been many changes within our company over the past two years to help streamline our operations and create a strong financial base for the future. Do you feel the changes that have been made have affected how you view our company in a negative way?

YES

NO

Next

Dealing with UI Events

What Events can be Hooked into?

There are a number of events that can be hooked into through scripting. A list of the key events for input devices are shown below.

Device	Event	Details
Mouse	<i>click</i>	Fires when the pointing device button is clicked over an element. A click is defined as a mousedown and mouseup over the same screen location.
	<i>dblclick</i>	Fires when the pointing device button is double clicked over an element
	<i>mousedown</i>	Fires when the pointing device button is pressed over an element
	<i>mouseup</i>	Fires when the pointing device button is released over an element
	<i>mouseover</i>	Fires when the pointing device is moved onto an element
	<i>mousemove</i>	Fires when the pointing device is moved while it is over an element
	<i>mouseout</i>	Fires when the pointing device is moved away from an element
Keyboard	<i>keydown</i>	Fires before keypress, when a key on the keyboard is pressed.
	<i>keypress</i>	Fires after keydown, when a key on the keyboard is pressed.
	<i>keyup</i>	Fires when a key on the keyboard is released

There are other events that relate to changes in the content of the survey page as follows:

Event	Details
<i>select</i>	Fires when a user selects some text in a text field, including input and textarea
<i>change</i>	Fires when a control loses the input focus and its value has been modified since gaining focus
<i>focus</i>	Fires when an element receives focus either via the pointing device or by tab navigation
<i>blur</i>	Fires when an element loses focus either via the pointing device or by tabbing navigation

How can Events be Used?

To use events, we need to do three things:

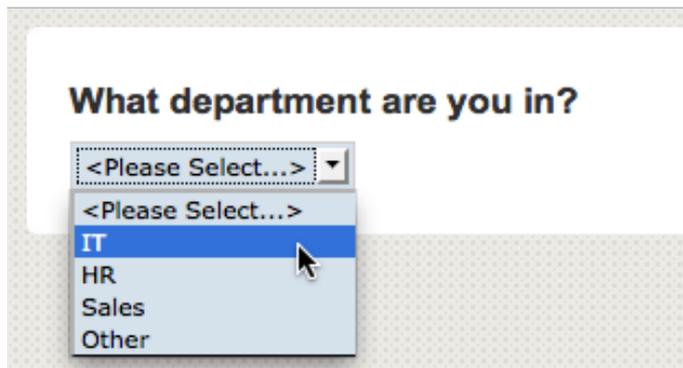
1. We need to have an HTML interface element to attach the event to - for example, a text field to check for changes
2. We need to indicate which Event (select, change, focus etc.) we want to listen out for
3. We need to script what needs to happen when the event is fired

Getting an HTML Interface Element

The act of “getting” an interface element differs depending upon what type of question you are scripting for. For simple questions that have only one input control - like text, drop-down and numeric question, you can get to the HTML Interface Element relatively easily.

Getting a Simple Interface Element

Let’s assume we have a drop-down list question with a unique code of DEPARTMENT.



In the script below, we are loading the HTML Element for this question.

```
// Get the Question
var oQuestion = wscScripting.getQuestionByDataPipingCode('DEPARTMENT');

if (oQuestion)
{
    // Get the HTML Element for this Question
    var HTMLElementForQuestion = $('#' + oQuestion.identity);
}
}
```

Getting Question Choice Interface Elements

Attaching an event script to a choice question can be very useful. The trick, however, is to deal with the fact that a choice question is has multiple interface elements - each of the choices. Let's assume we have a drop-down list question with a unique code of DEPARTMENTCHOICE.



In the script below, we are loading the HTML Elements for each of the choices in this question.

```
// Get the Question
var oQ = wscScripting.getQuestionByDataPipingCode('DEPARTMENTCHOICE');

if (oQ)
{
    // Get the HTML Elements for each choice
    var ChoiceButtonIT = $('#' + oQ.identity + '_' + oQ.choices[0].identity);
    var ChoiceButtonHR = $('#' + oQ.identity + '_' + oQ.choices[1].identity);
    var ChoiceButtonSales = $('#' + oQ.identity + '_' + oQ.choices[2].identity);
    var ChoiceButtonOther = $('#' + oQ.identity + '_' + oQ.choices[3].identity);
}
}
```

Note that access to choices is zero-based. Therefore, to access the first choice, we actually get:

```
oQ.choices[0].identity
```

Complex Interface Elements

Getting an interface element to attach an event to can be “tricky” if you are dealing with more complex question types. Let’s assume we have a constant sum question with a unique code of DEPARTMENTSUM.

How many hours a week do you spend dealing with people from these departments?

	Hours
IT	<input type="text" value="6"/>
HR	<input type="text" value="10"/>
Sales	<input type="text" value="15"/>
Other	<input type="text" value="6"/>
Total	37 hours

A Constant Sum question is effectively a matrix with one column (“hours” in our example), and a number of rows. If we wanted to attach an event to the “other” numeric element, we’d need to first get to the row. The easiest way to get a row is to search for a particular row tag and value:

```
// Get the Question
var oQ = wscScripting.getQuestionByDataPipingCode('DEPARTMENTSUM');

if (oQ)
{
    // Get the Row by looking for a specific tag with a specific value
    var rowObject = wscScripting.getRowByTagValue(constantSum, 'ROWTAG', 'OtherRow');

    if (rowObject)
    {
        // Get the HTML Element for the numeric input box
        // We need to refer to both the row, and the column (choice)
        var inputBox = $('#'+oQ.identity+'_'+rowObject.identity+'_'+oQ.choices[0].identity);
    }
}
```

Attaching an Event

Once we have an HTML element, attaching an event is quite trivial. The appropriate event is “bound” to the HTML element, and a function is run whenever that event occurs.

```

// Get the Question
var oQuestion = wscScripting.getQuestionByDataPipingCode('DEPARTMENT');

if (oQuestion)
{

    // Get the HTML Element for this Question
    var HTMLElementForQuestion = $('#' + oQuestion.identity);

    // Do something when the value changes
    $(HTMLElementForQuestion).bind('change',
        function()
        {

            // Script to run goes here

        });
}

args.isValid = true;

```

UI Event Example

Let's consider a complete example for one of the question types we have discussed earlier in this section.

Drop-down List with “Other”

Choice questions that are shown as radio button or check boxes include a text box for other in the question. Drop-down lists don't have this capability. We can add this using scripting, however. All we need is an “Other” text question that is hidden when Other is selected.

Before we set up our script, the questions would look as follows:



The screenshot shows a user interface with two distinct sections. The top section contains the question "What department are you in?" followed by a dropdown menu with the text "<Please Select...>". The bottom section contains the question "Other, Please Specify." followed by a text input field.

We only want the “Other Specify” question to show if “other” is selected - this is what the script needs to do for us. Our two questions have the following unique codes:

DEPARTMENT

OTHERDEPARTMENT

```

// Get the Drop-down
var oDD = wscScripting.getQuestionByDataPipingCode('DEPARTMENT');
// Get the "Other" text question
var oOther = wscScripting.getQuestionByDataPipingCode('OTHERDEPARTMENT');

if (oDD && oOther)
{ // We want to only show the other question if the DD = 4 (Other)

    // Get the HTML Element for Drop Down List
    var HTMLElementForDD = $('#' + oDD.identity);
    // Get the HTML Element for Other Specify box
    var HTMLElementForOther = $('#' + oOther.identity);

    // What Choice is currently selected?
    var aSelectedItem = wscScripting.getSelectedChoices(oDD);

    if (aSelectedItem)
    {
        if (aSelectedItem.value != 4) // Other is not selected
        {
            // Hide the other text question on the load
            $('#' + oOther.containerName).hide();
        }
    }

    // Do something when the value of the Drop-Down changes
    $(HTMLElementForDD).bind('change',
        function() {

            var LocalHTMLElementForDD = $('#' + oDD.identity);
            var LocalHTMLElementForOther = $('#' + oOther.containerName);
            // Get the "Other" choice for testing
            var oChoice = wscScripting.getChoiceByValue(oDD, 4);

            if (LocalHTMLElementForDD.val() == oChoice.identity)
            { // They chose the other choice - show "Other"
                LocalHTMLElementForOther.fadeIn('slow', function() {});
            }
            else
            { // They didn't choose the other choice
                LocalHTMLElementForOther.fadeOut('slow', function() {});
            }

        });

    });

args.isValid = true;

```

Scripting #101:

Ordering of Pages & Choices

All surveys follow a particular “flow”. At it’s simplest, this is just going from one page to the next, and one choice to the next, in the order a survey was created.

Sometimes, however, survey flow needs to be managed in a very specific way that varies from respondent to respondent. This is when scripting needs to be used.

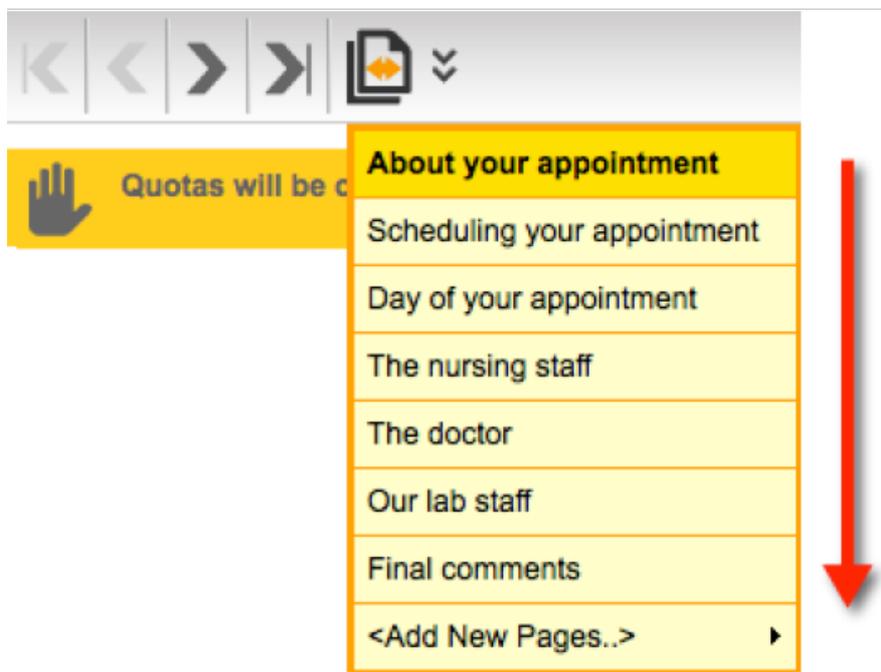


Page Ordering in the Designer

Before considering how page ordering may be managed through scripting, let's first consider how page ordering can be achieved without scripting.

Basic Page Order

Basic page ordering is determined by the order in which they have been added to a survey in the content manager.



When new pages are added to a survey, they can be added before or after an existing page by pressing the appropriate button at the top or bottom of the existing page.



This is of course the way page ordering works, even on the most basic survey systems.

Randomization of Pages

If everyone sees pages in the same order, there can be some unforeseen effects on responses including:

1. Better quality responses for pages earlier in long surveys - later pages suffer from respondent fatigue
2. Responses later in a survey being affected by questions seen by the respondent earlier in a survey - they make react differently because of what they have previously seen

So, if ordering can change resulting data, how can we minimize these problems?

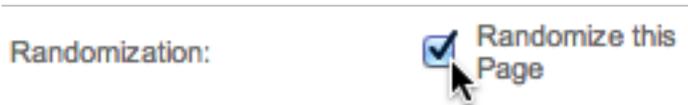
The only effective way to minimize how page ordering affects responses is to spread the possible ordering bias evenly. Specifically - page order needs to be randomized.

To use standard page randomization in Web Survey Creator, you need to:

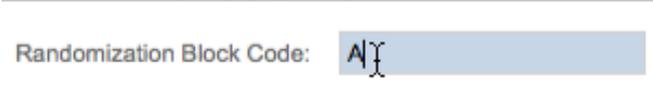
1. Edit the first page you want to randomize



2. Check the randomization check box



3. If you want to group pages together in the random order, enter an optional Block Code

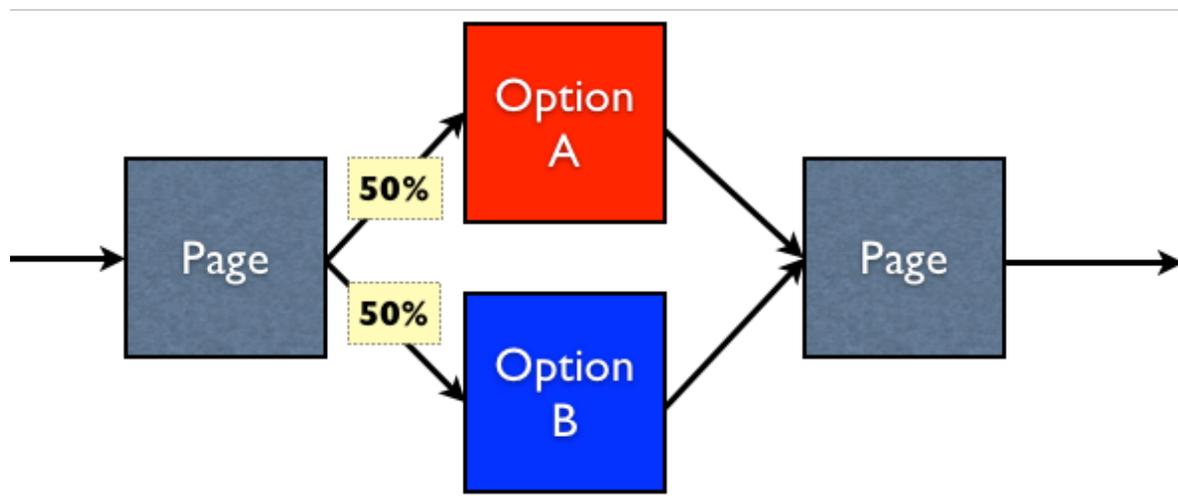


4. Save the page
5. Repeat for each subsequent page you wish to randomize.

Another way to randomize - A/B Testing

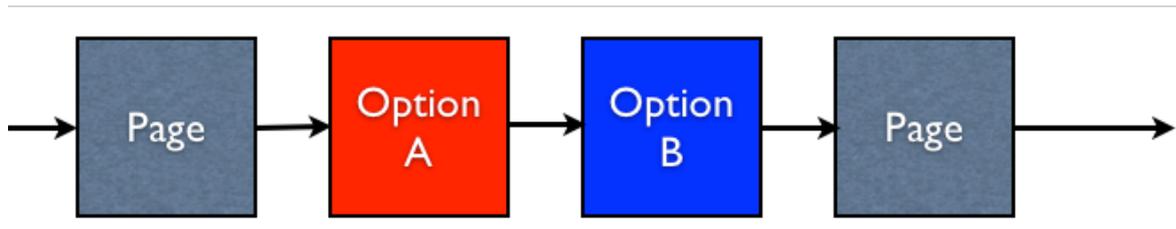
Users of A/B testing will distribute multiple samples of a test to see which single variable is most effective in increasing a response rate or other desired outcome. The test, in order to be effective, must reach an audience of a sufficient size that there is a reasonable chance of detecting a meaningful difference between the control and other tactics.

Web surveys are a great candidate to use A/B testing, since gaining access to a large audience is relatively easy. A simple example of an A/B test is shown below.



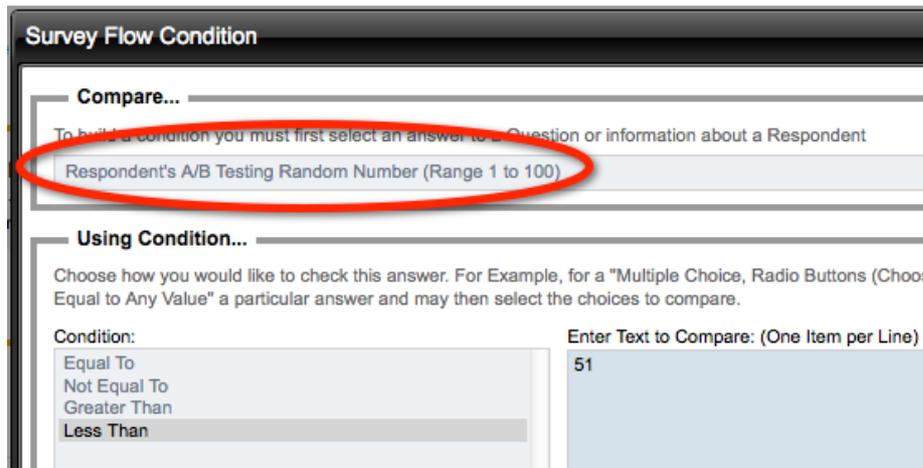
What we want is 50% of respondents to go to the page Option A, and 50% to go to the page Option B. Achieving this result is another example of randomization.

As soon as a respondent begins a response in Web Survey Creator, they are allocated a random number. This number can be used for various things - one of which is A/B testing. Setting up an A/B test is actually done through flow control. The true structure of a survey that has the A/B test above would be:



To create our A/B test, we want to hide the Option A page 50% of the time, and hide the Option B page 50% of the time.

Hiding the option A page would require the following flow:



We use the A/B testing Random Number for the Respondent (which is always a number between 1 and 100) to do the flow. We want to hide the Option A page if the number is less than 51 (i.e. the number is between 1 and 50).

The hiding of the option B page would be the opposite test - hide the page when the A/B testing Random Number is greater than 50 (i.e. the number is between 51 and 100).

Of course this methodology wouldn't just apply to a two page test - you could have up to 100 options that are randomly chosen between, since the AB testing Random Number is equal to 1 of a possible 100 values.

Page Ordering through Scripting

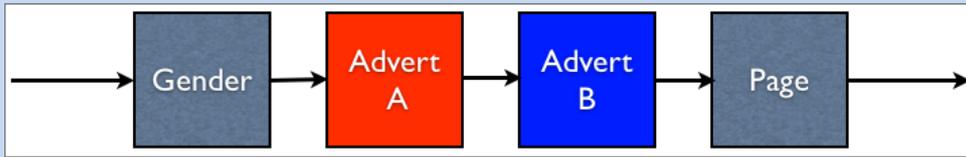
Randomization is great for spreading the effects of bias due to page order. It does have one limitation though, that may cause issues in circumstances...

Randomization cannot be predicted - the order of pages a particular respondent will see is determined at the time they enter their response.

This is great in most circumstances, but what happens if I want to manipulate the order in a known way? To achieve this, we would need to modify the page order through scripting.

Let's consider an example...

We want to create a survey that includes two advertisements - each of their own page. The basic flow of the survey will be as follows:



Our client wants us to show *Advert A* first for all men and *Advert B* first for all women. The order of the pages is therefore determined by the answer to a particular question, rather than being completely random.

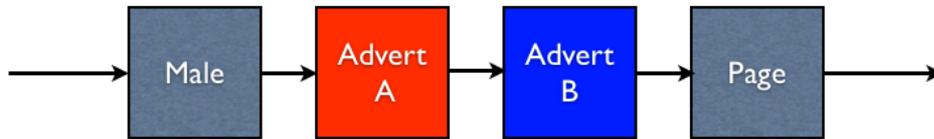
What is needed for Page Order Scripting?

If you want to manage page order through scripting, there are a couple of things that need to be set up correctly as follows:

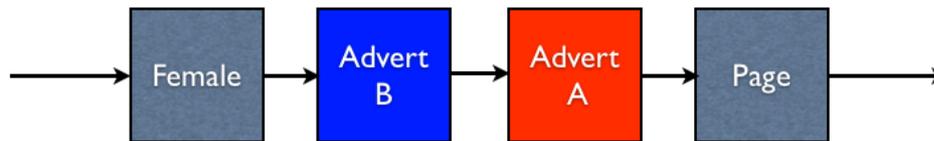
1. **Name your pages** so they can be referred to by name in the script. In our example, we will edit the page names and make them:
 - a. Gender
 - b. Advert A
 - c. Advert B
 - d. Next Page
2. **Make at least one page in the survey randomized** so the system knows to include data in pages to allow script to access page order. You will be able to move any pages in a survey, as long as at least one is randomized.
3. **(Optionally) place individual randomization block codes on pages** if you want to access them by block code (rather than page name).
4. **Add the reordering script** to a page that comes before the pages to be reordered (it will be run when the next button is pressed, and pages will be reordered before the survey moves forward).

Our Example: Ordering Pages based on Gender

In our previous example, we want to set up our page ordering so that men see the pages in this order:

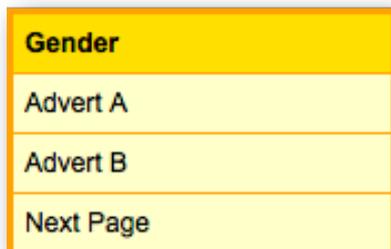


and women see pages in this order:



In order to achieve this, we do the following setup in our survey:

1. Name the pages of our survey “Gender”, “Advert A”, “Advert B” and “Next Page”



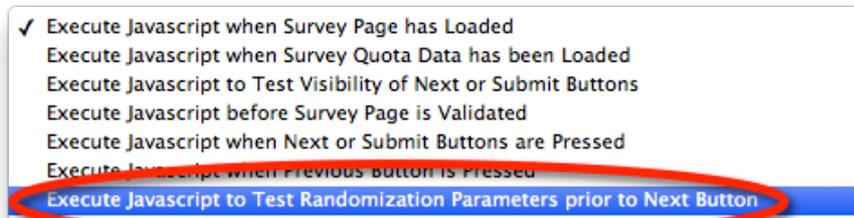
2. Add the access code “GENDER” to our gender question.

Question Access Code (Optional) [Used for Data Piping, SPSS, etc]:

3. Set the advertisement pages to random, and give them both a block code of “ADVERTS”. We are doing this so we can get to them in the script through their block code.

Randomization: Randomize this Page Randomization Block Code:

4. Create a **Randomization script**.



Web Survey Creator will provide all the data you need on a page for scripted changing of page orders as long as two things are true - at least one page is randomized in the survey **AND** a test randomization parameter script exists on the page. If either these things is false, no ordering is possible.

OK, everything is set up on our survey to allow us to manage the ordering of the pages. Let's go through the script we need to create.

The first thing we will need to do in the script is get to the value of the gender question:

```
// Did we pick a male or female?
var bMale = false;
var oGender = wscScripting.getQuestionByDataPipingCode('GENDER');
if (oGender != null) {
    var aSelected = wscScripting.getSelectedChoices(oGender);
    if (aSelected != null) {
        bMale = (aSelected[0].value == 1);
    }
}
```

We now know whether the respondent is male or not. The next step is to get the page items in an array for the survey, and work out where the advertisement pages are in this array.

```
// Get the page order
var aPages = wscScripting.getPageRandomizationItems();
if (aPages != null) {
    // Find the pages that are blocked as GENDER...
    // We need to changed their order dependant on the gender
    var nOrdinal = -1;
    for(var nPage = 0;nPage<aPages.length;nPage++) {
        if (aPages[nPage].blockCode == 'ADVERTS') {
            // this is the right block
            nOrdinal = nPage;
            break;
        }
    }
}
```

We want to...	Key script Fragment...
Get the pages for the survey	wscScripting.getPageRandomizationItems

Through this script, we have determined the position in the pages array that contains the first of our advertisements. We know that the page after this page will be the second advertisement.

This example shows a clever use of **Block Codes** on pages. The two pages we want to swap around have been given the same block code:

ADVERTS

This means in our script we can just work through pages and look for the block code "ADVERTS". As soon as we find a page with this block code, we know we have found the *first of our two advertisement pages*. If we didn't use the block code, we'd have to check for both of our pages by name as we moved through the array of pages, because we wouldn't know which page was shown first.

All that is left to do is make sure the advertisements are in the correct order. The full code for the script (including the ordering) is shown below:

```

// Did we pick a male or female?
var bMale = false;
var oGender = wscScripting.getQuestionByDataPipingCode('GENDER');
if (oGender != null) {
    var aSelected = wscScripting.getSelectedChoices(oGender);
    if (aSelected != null) {
        bMale = (aSelected[0].value == 1);
    }
}
// Get the page order
var aPages = wscScripting.getPageRandomizationItems();
if (aPages != null) {
    // Find the pages that are blocked as GENDER...
    // We need to changed their order dependant on the gender
    var nOrdinal = -1;
    for(var nPage = 0;nPage<aPages.length;nPage++) {
        if (aPages[nPage].blockCode == 'ADVERTS') {
            // this is the right block
            nOrdinal = nPage;
            break;
        }
    }
    // Did we correctly find this page?
    if (nOrdinal > -1) {
        // if bMale then should be Advert A first
        // if !bMale then should be Advert B first
        // get them so we can flip them
        var oPage1 = aPages[nOrdinal];
        var oPage2 = aPages[nOrdinal + 1];
        // we might already have done this so check the title to be sure
        var sTitle = oPage1.title;
        if (bMale && sTitle == 'Advert B') {
            // Should be Advert A so flip them
            aPages[nOrdinal] = oPage2;
            aPages[nOrdinal + 1] = oPage1;
            // update them
            wscScripting.setPageRandomizationItems(aPages);
        }
        else if (!bMale && sTitle == 'Advert A') {
            // Should be Advert B so flip them
            aPages[nOrdinal] = oPage2;
            aPages[nOrdinal + 1] = oPage1;
            // update them
            wscScripting.setPageRandomizationItems(aPages);
        }
    }
}
args.isValid = true;

```

Note how we save down the page items at the end of the script once they are ordered in the way we want.

We want to...	Key script Fragment...
Save the pages in order back to the survey	wscScripting.setPageRandomizationItems

Page Ordering in a Nutshell

As can be seen from the previous example, a page ordering script has 3 distinct parts to it:

Step 1: Get the current page order

The script gets the current page objects into an array using *getPageRandomizationItems*.

```
// Get the page order
var aPages = wscScripting.getPageRandomizationItems();
```

Step 2: Alter the Page Order in the array from step 1

This is the heart of the script - you do whatever is needed to move pages around in the array. As we can see from the previous example, getting page objects from the array is easy:

```
var oPage1 = aPages[nOrdinal];
var oPage2 = aPages[nOrdinal + 1];
```

As is putting them back into the array:

```
aPages[nOrdinal] = oPage2;
aPages[nOrdinal + 1] = oPage1;
```

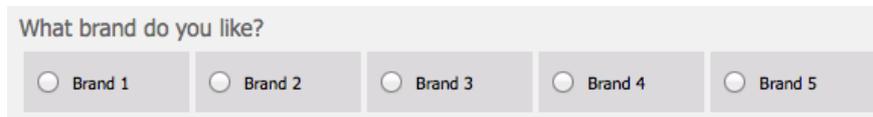
Step 3: Update the Page Ordering from the Array

Once you have finished manipulating the array, it can be used to update the page ordering for the survey.

```
wscScripting.setPageRandomizationItems(aPages);
```

Ordering Choices

Choice questions are one of the simplest, and most used question types in Web Survey Creator. An example of a choice question is shown below:

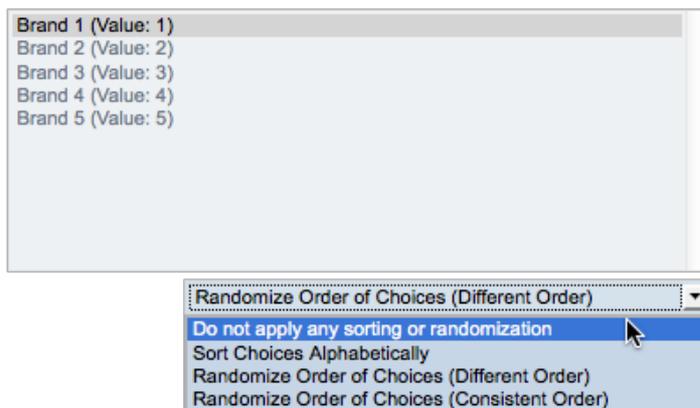


What brand do you like?

Brand 1 Brand 2 Brand 3 Brand 4 Brand 5

Standard Ordering of Choices

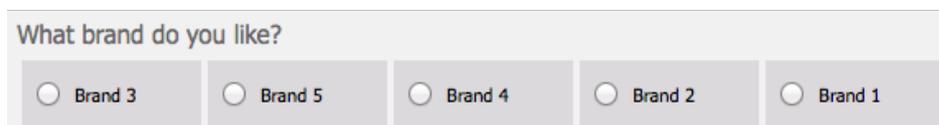
The vast majority of choice questions are ordered the same way as they appear in the content manager. It is also possible to choose other ordering options when adding or editing a choice question as follows:



Brand 1 (Value: 1)
Brand 2 (Value: 2)
Brand 3 (Value: 3)
Brand 4 (Value: 4)
Brand 5 (Value: 5)

Randomize Order of Choices (Different Order)
Do not apply any sorting or randomization
Sort Choices Alphabetically
Randomize Order of Choices (Different Order)
Randomize Order of Choices (Consistent Order)

Here is what our example question might look like, if we chose to randomize the order of choices:



What brand do you like?

Brand 3 Brand 5 Brand 4 Brand 2 Brand 1

While alphabetical sorting and randomization can be very useful, sometimes you want to set up an exact order of choices. This is when scripting needs to be used to manage choices.

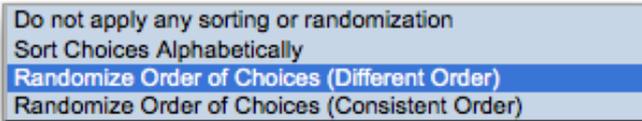
Setting up for Scripted Choice Ordering

All question types that have choices can have the ordering of those choice managed through scripting. For a question to be available in scripting, the following setup must be done:

1. **An access code must set up on the question** so that the question can be referred to in script.

Question Access Code (Optional) [Used for Data Piping, SPSS, etc]: BRANDS

2. **The choices must be randomized** by selecting one of the two randomization choices on the question.



3. **Question must be made available for scripting** - if this is not ticked, the system will not put the appropriate data into the page for choice ordering to be modifiable.



Example: Scripted Choice Ordering

There are any number of reasons you may want to reorder choices. For the purposes of demonstration, we are going to use a contrived, but straight-forward example - we will take a randomized set of choices, and in script sort them alphabetically.

Choice Questions

Let's start with a question that is randomized and has the following values:

What is your favorite fruit?

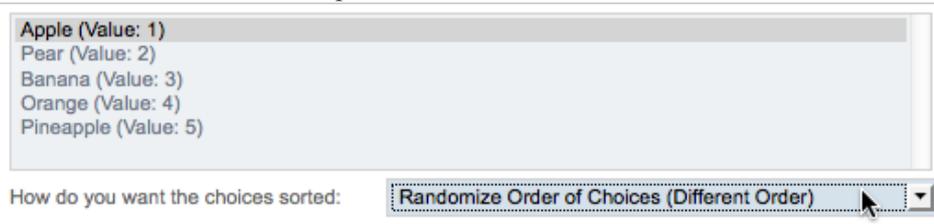
- Apple
- Pear
- Banana
- Orange
- Pineapple

To script this question so that the values are alphabetical, we first need to set the question up so it can be used with scripting:

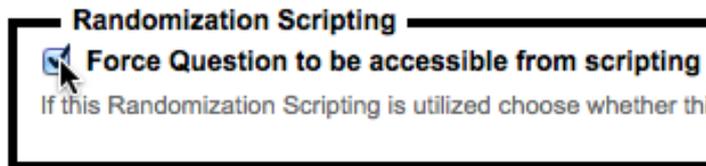
1. **Set Access Code** on the question so it can be accessed through scripting

Question Access Code (Optional) [Used for Data Piping, SPSS, etc]:

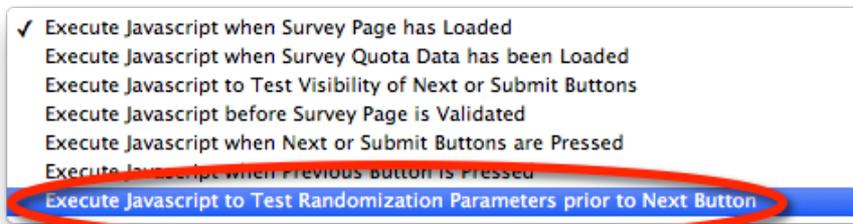
2. Use **randomization** for the question choices



3. Make the question **accessible to scripting**



We then need to write a script that does the re-ordering of the choices. The script needs to be set up as a test of randomization parameters (similar to the scripts that are used for page ordering):



The script to change the order of choice items in a question must be created on a page prior to the page containing the actual question. Trying to change the ordering on the same page as the question will not work - the script would be run too late.

The first thing we would need to do in our script is load the details for our question:

```
oQuestion = wscScripting.getQuestionByDataPipingCode('SINGLE');  
if (oQuestion) {  
    aRandom = wscScripting.getChoiceRandomizationItems(oQuestion, 0);  
}
```

We want to...	Key script Fragment...
Get the choices for a question	wscScripting.getChoiceRandomizationItems

For the purposes of our demonstration, we are going to go through the choice objects, and set up the sort order so that the choices are in alphabetical order.

We achieve this by updating "sortorder" with the actual text of each of the choices, and then re-sorting the array.

```

// Get the title from the choice
for(var nChoice = 0;nChoice < oQuestion.choices.length;nChoice++) {
  var oChoice = oQuestion.choices[nChoice];
  var sText = oChoice.text;
  var sIdentity = oChoice.identity;
  for(var nRandom = 0;nRandom < aRandom.length;nRandom++) {
    if (aRandom[nRandom].identity.toLowerCase() == sIdentity) {
      aRandom[nRandom].sortOrder = sText.toLowerCase();
      break;
    }
  }
}
// Sort it
aRandom.sort(function(a,b) {
  if (a.sortOrder < b.sortOrder) { return -1 }
  if (b.sortOrder < a.sortOrder) { return 1 }
  if (b.sortOrder == a.sortOrder) { return 0 }
} );

```

Once we have re-sorted our array, we can save it back to the survey.

```

// Update the randoms for updating on next
bSuccess = wscScripting.setChoiceRandomizationItems(oQuestion, aRandom, 0);

```

We want to...	Key script Fragment...
Save the choices for a question back to the survey	wscScripting.setChoiceRandomizationItems

```

var oQuestion;
var aRandom;
var bSuccess;

// Single Selection List
oQuestion = wscScripting.getQuestionByDataPipingCode('SINGLE');
if (oQuestion) {
    aRandom = wscScripting.getChoiceRandomizationItems(oQuestion, 0);

    if (aRandom) {

        // Get the title from the choice
        for(var nChoice = 0;nChoice < oQuestion.choices.length;nChoice++) {
            var oChoice = oQuestion.choices[nChoice];
            var sText = oChoice.text;
            var sIdentity = oChoice.identity;
            for(var nRandom = 0;nRandom < aRandom.length;nRandom++) {
                if (aRandom[nRandom].identity.toLowerCase() == sIdentity) {
                    aRandom[nRandom].sortOrder = sText.toLowerCase();
                    break;
                }
            }
        }

        // Sort it
        aRandom.sort(function(a,b) {
            if (a.sortOrder < b.sortOrder) { return -1 }
            if (b.sortOrder < a.sortOrder) { return 1 }
            if (b.sortOrder == a.sortOrder) { return 0 }
        });

        // Update the randoms for updating on next
        bSuccess = wscScripting.setChoiceRandomizationItems(oQuestion, aRandom, 0);
    }
}

args.isValid = true;

```

Once this script is run, the choices in the question will appear in alphabetical order as shown here:

What is your favorite fruit?

Apple

Banana

Orange

Pear

Pineapple

Single Range Matrix

Let's now consider another "choice" question - a single range matrix. For this example, let's use a matrix that looks as follows:

What is your favorite fruit for the following?					
	Apple	Pear	Banana	Orange	Pineapple
Cakes	<input type="radio"/>				
Juice Drinking	<input type="radio"/>				
Late night snack	<input type="radio"/>				
Because you are hungry	<input type="radio"/>				

Setting up this question so that we can manipulate choice order (i.e. column order) is exactly like a standard choice question. We need to perform the same setup:

1. **Set Access Code** on the question so it can be accessed through scripting. In this case we will give the question a code of "MATRIXSINGLE"
2. **Use randomization** for the question choices
3. Make the question **accessible to scripting**

The script will *exactly* match the script for the simple choice question, with one simple change - the code to access the question:

```
// Single Matrix  
oQuestion = wscScripting.getQuestionByDataPipingCode('MATRIXSINGLE');
```

Accessing and modifying the choices in questions work in a similar way, as can be seen by the scripts used for single choice questions, and single range matrixes - they are essentially identical. This is also true for other questions that have choices - such as ranking questions.

The full script for sorting the matrix rows into alphabetical order is shown here:

```

var oQuestion;
var aRandom;
var bSuccess;

// Single Matrix
oQuestion = wscScripting.getQuestionByDataPipingCode('MATRIXSINGLE');
if (oQuestion) {
  aRandom = wscScripting.getChoiceRandomizationItems(oQuestion, 0);
  if (aRandom) {

    // Get the title from the choice
    for(var nChoice = 0;nChoice < oQuestion.choices.length;nChoice++) {
      var oChoice = oQuestion.choices[nChoice];
      var sText = oChoice.text;
      var sIdentity = oChoice.identity;
      for(var nRandom = 0;nRandom < aRandom.length;nRandom++) {
        if (aRandom[nRandom].identity.toLowerCase() == sIdentity) {
          aRandom[nRandom].sortOrder = sText.toLowerCase();
          break;
        }
      }
    }
    // Just sort by the natural order rather than random
    aRandom.sort(function(a,b) {
      if (a.sortOrder < b.sortOrder) { return -1 }
      if (b.sortOrder < a.sortOrder) { return 1 }
      if (b.sortOrder == a.sortOrder) { return 0 }
    });

    // Update the randoms for updating on next
    bSuccess = wscScripting.setChoiceRandomizationItems(oQuestion, aRandom, 0);
  }
}
args.isValid = true;

```

The columns will appear in alphabetical order:

What is your favorite fruit for the following?

	Apple	Banana	Orange	Pear	Pineapple
Cakes	<input type="radio"/>				

Choices are the same for all question types - but matrix questions also have **rows**. Managing rows is similar to choices - you just need to use the right functions that relate to rows...

We want to...	Key script Fragment...
Get the rows for a question	wscScripting.getRowRandomizationItems
Save the rows for a question back to the survey	wscScripting.setRowRandomizationItems

This is the script we use to sort rows:

```
var oQuestion;
var aRandom;
var bSuccess;

// Single Matrix
oQuestion = wscScripting.getQuestionByDataPipingCode('MATRIXSINGLE');
if (oQuestion) {

    aRandom = wscScripting.getRowRandomizationItems(oQuestion);
    if (aRandom) {
        // Get the title from the row
        for(var nRow = 0;nRow < oQuestion.rows.length;nRow++) {
            var oRow = oQuestion.rows[nRow];
            var sText = oRow.text;
            var sIdentity = oRow.identity;
            for(var nRandom = 0;nRandom < aRandom.length;nRandom++) {
                if (aRandom[nRandom].identity.toLowerCase() == sIdentity) {
                    aRandom[nRandom].sortOrder = sText.toLowerCase();
                    break;
                }
            }
        }

        // Sort it
        aRandom.sort(function(a,b) {
            if (a.sortOrder < b.sortOrder) { return -1 }
            if (b.sortOrder < a.sortOrder) { return 1 }
            if (b.sortOrder == a.sortOrder) { return 0 }
        });

        // Update the randoms for updating on next
        bSuccess = wscScripting.setRowRandomizationItems(oQuestion, aRandom);
    }
}

args.isValid = true;
```

The question will then look as follows - with both rows and columns in alphabetical order:

What is your favorite fruit for the following?

	Apple	Banana	Orange	Pear	Pineapple
Because you are hungry	<input type="radio"/>				
Cakes	<input type="radio"/>				
Juice Drinking	<input type="radio"/>				
Late night snack	<input type="radio"/>				

Dual Range Matrix

We have already seen how choices can be retrieved and saved for a single range matrix. We get the choices as follows:

```
aRandom = wscScripting.getChoiceRandomizationItems(oQuestion, 0);
```

And to save choices we do the following:

```
bSuccess = wscScripting.setChoiceRandomizationItems(oQuestion, aRandom, 0);
```

So how do we get to the second matrix in a dual range matrix?

The answer to this can be seen in the two lines above. If you are dealing with any question type except a dual range matrix, the “range” that will be dealt with is the first matrix - designated by a zero (0) in the function call:

```
wscScripting.getChoiceRandomizationItems(oQuestion, 0);
wscScripting.setChoiceRandomizationItems(oQuestion, aRandom, 0);
```

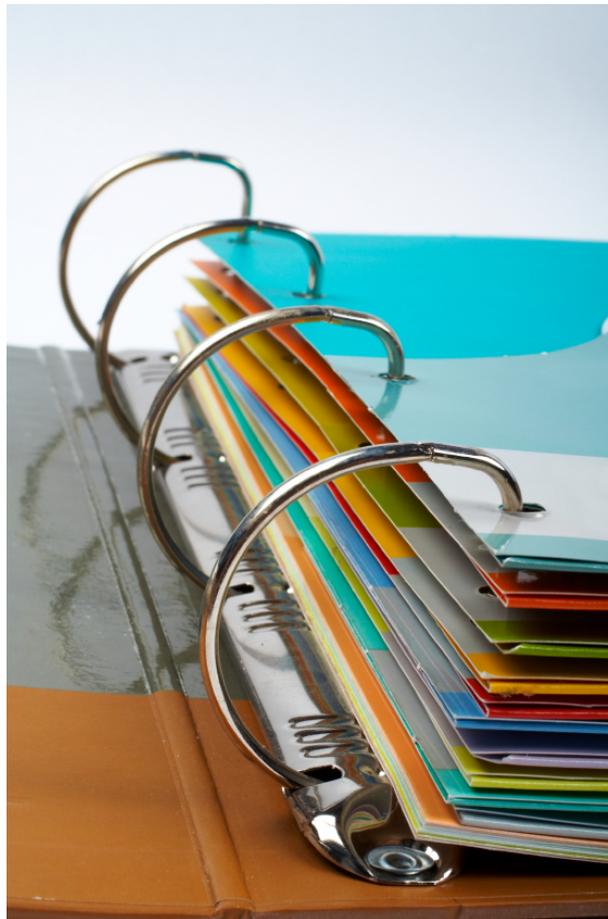
For a dual range matrix, if we want to load and save the choices from the second matrix, we enter one (1) in the function call:

```
wscScripting.getChoiceRandomizationItems(oQuestion, 1);
wscScripting.setChoiceRandomizationItems(oQuestion, aRandom, 1);
```

Scripting Reference

Web Survey Creator provides specific objects and methods for use in scripting.

This chapter is a reference guide for these objects and methods.



Scripting Objects

When a custom JavaScript is executed you will have access to two objects. These objects allow you access to the questions that are exposed on the current page and additional help methods that can help you to perform various tasks.

args

wscScripting

args

args contains a single item isValid that can be used to set the status of an event. This is particularly relevant for confirming to the event engine that you wish to continue the current process. For example, you must set the value to true on Next or Previous Button events or those processes halted and will not continue.

Property:	isValid
Return Value:	boolean - Is the current process Valid
Example:	<pre>var isOkay = true; if (isOkay) { // All my changes allow me to continue args.isValid = true; }</pre>

wscScripting

The following methods available in the wscScripting object. Some methods contain a method and an identical method post-fixed with the number 2. These methods are used where the question has two (2) choice ranges.

For example, Dual Range Matrix questions consist of a Primary Range and a Secondary Range.

In these circumstances the Secondary Range can be utilized by using the methods with a post-fix of 2. For example. getChoiceByValue2(question, value). In this document methods post-fixed with a number of 2 will be documented only in their primary method. Each method explanation will denote if the method has a second range capability.

A listing of the methods available for the wscScripting object are shown below. Detailed explanations are provided in the next section.

clearValidation(question)

derankChoice(question, choice)

deselectChoice(question, choice)

deselectChoice2(question, choice)

deselectChoiceByValue(question, value)
deselectChoiceByValue2(question, value)
getChoiceRandomizationItems(question, matrixnumber)
getRowRandomizationItems(question)
getPageRandomizationItems()
deselectMatrixChoice(question, choice, row)
disableQuestion(question)
enableQuestion(question)
getABTesting()
getBrowserData()
getChoiceByTagValue(question, tagName, value)
getChoiceByTagValue2(question, tagName, value)
getChoiceByValue(question, value)
getChoiceByValue2(question, value)
getDateStringFromDate(date)
getDirection()
getDisplayType()
getDistribution()
getElementById(id)
getEventData(name)
getLanguageId()
getQuestionByDataPipingCode(dataPipingCode)
getQuestionByIdentity(identity)
getQuotaByCode(code)
getQuotaByIdentity(identity)
getRecallCode()
getRowByTagValue(question, tagName, value)

`getSelectedChoices(question)`
`getSelectedChoices2(question)`
`getSelectedMatrixChoices(question, row)`
`getSelectedMatrixChoices2(question, row)`
`getSelectedRanks(question)`
`getSubstringLeft(str, n)`
`getSubstringRight(str, n)`
`getTrimString(str)`
`getValidators(question)`
`getValue(question)`
`hideElement(element)`
`isAnyChoiceSelected(question, choices)`
`isAnyChoiceSelected2(question, choices)`
`isAnyChoiceSelectedByValue(question, values)`
`isAnyChoiceSelectedByValue2(question, values)`
`isAnyMatrixChoiceSelected(question, choices, row)`
`isAnyMatrixChoiceSelected2(question, choices, row)`
`isAnyMatrixChoiceSelectedByValue(question, values, row)`
`isAnyMatrixChoiceSelectedByValue2(question, values, row)`
`isChoiceSelected(question, choice)`
`isChoiceSelected2(question, choice)`
`isChoiceSelectedByValue(question, value)`
`isChoiceSelectedByValue2(question, value)`
`isMatrixChoiceSelected(question, choice, row)`
`isMatrixChoiceSelected2(question, choice, row)`
`isMatrixChoiceSelectedByValue(question, value, row)`
`isMatrixChoiceSelectedByValue2(question, value, row)`

`rankChoice(question, choice, rank)`
`selectChoice(question, choice)`
`selectChoice2(question, choice)`
`setChoiceRandomizationItems(question, choices, matrixnumber)`
`setRowRandomizationItems(question, rows)`
`setPageRandomizationItems(pages)`
`selectChoiceByValue(question, value)`
`selectChoiceByValue2(question, value)`
`selectMatrixChoice(question, choice, row)`
`selectMatrixChoice2(question, choice, row)`
`selectMatrixChoiceByValue(question, value, row)`
`selectMatrixChoiceByValue2(question, value, row)`
`setEventData(name, value)`
`setValidation(question, text)`
`setValue(question, value)`
`showElement(element)`

Function Reference

Method: **clearValidation(question)**

Parameters: question object - object of the question

Return Value: Nil

Example:

```
var question =
wscScripting.getQuestionByDataPipingCode('mydatapipingcode');
if (question) {
    wscScripting.clearValidation(question);
}
```

Method: **derankChoice(question, choice)**

Parameters: question object - object of the question

choice object - object of a choice

Return Value: Nil

Example:

```
var question =
wscScripting.getQuestionByDataPipingCode('mydatapipingcode');
if (question) {
    var choice = wscScripting.getChoiceByValue(question, 1);
    if (choice) {

        wscScripting.derankChoice(question, choice);
    }
}
```

Method: **deselectChoice(question, choice)**

second range

Parameters: question object - object of the question

choice object - object of a choice

Return Value: boolean - confirmation that the choice was deselected

Example:

```
var question =
wscScripting.getQuestionByDataPipingCode('mydatapipingcode');
if (question) {
    var choice = wscScripting.getChoiceByValue(question, 1);
    if (choice) {

        var isUnselected =
wscScripting.deselectChoice(question, choice);
    }
}
```

Method: **deselectChoiceByValue(question, number)** **second range**
Parameters: question object - object of the question
number - value of the question choice to check
Return Value: boolean - confirmation that the choice was deselected
Example:

```
var question =  
wscScripting.getQuestionByDataPipingCode('mydatapipingcode');  
if (question) {  
  
    var isUnselected =  
wscScripting.deselectChoiceByValue(question, 1);  
}
```

Method: **deselectMatrixChoice(question, choice, row)** **second range**
Parameters: question object - object of the question
choice object - object of a choice
row object - object of a row
Return Value: boolean - confirmation that the choice was deselected
Example:

```
var question =  
wscScripting.getQuestionByDataPipingCode('mydatapipingcode');  
if (question) {  
    var row = wscScripting.getRowByTagValue(question, 'bankcode',  
'AMER');  
    if (row) {  
  
        var choice = wscScripting.getChoiceByValue(question,  
1);  
        if (choice) {  
  
            var isUnselected =  
wscScripting.deselectMatrixChoice(question, choice, row);  
        }  
    }  
}
```

Method: **disableQuestion(question)**
Parameters: question object - object of the question
Return Value: boolean - confirmation that the choice was disabled
Example:

```
var question =  
wscScripting.getQuestionByDataPipingCode('mydatapipingcode');  
if (question) {  
    var isDisabled = wscScripting.disableQuestion(question);  
}
```

Method: **enableQuestion(question)**
Parameters: question object - object of the question
Return Value: boolean - confirmation that the choice was enabled
Example:

```
var question =  
wscScripting.getQuestionByDataPipingCode('mydatapipingcode');  
if (question) {  
    var isDisabled = wscScripting.enableQuestion(question);  
}
```

Method: **getABTesting()**
Parameters: Nil
Return Value: integer - Value (range 1..100) of for use by AB Testing
Example:

```
var ABTest = wscScripting.getABTesting();  
  
if (ABTest <= 50) { // Split 50:50  
  
}
```

Method: **getBrowserData()**
Parameters: Nil
Return Value: object - Browser Object

```
o.browser = Browser Name  
o.version = Browser Version  
o.OS = Operating System  
Example: var browser = wscScripting.getBrowserData();  
  
if (browser.OS == 'Windows') { // The respondent is on a  
Windows computer  
  
}
```

Method: **getChoiceByTagValue(question, tagName, value)** **second range**
Parameters: question object - object of the question
string - name of the tag to search for
string - value of the tag being searched
Return Value: object or undefined

```
Example: var question =  
wscScripting.getQuestionByDataPipingCode('mydatapipingcode');  
if (question) {  
  
    var choice = wscScripting.getChoiceByTagValue(question,  
'position', 'Manager');  
    if (choice) {  
  
        // I can do something with this choice  
    }  
}
```

Method: **getChoiceByValue(question, value)** second range
Parameters: question object - object of the question
number - value of the question choice to retrieve
Return Value: object or undefined
Example:

```
var question =
wscScripting.getQuestionByDataPipingCode('mydatapipingcode');
if (question) {

    var choice = wscScripting.getChoiceByValue(question, 1);
    if (choice) {

        // I can do something with this choice
    }
}
```

Method: **getDateStringFromDate(date)**
Parameters: Date = Value of Date type to be converted to a string in format usable by WSC
Return Value: string = Newly created string in format of YYYY.MM.DD.HH.mm
Example:

```
var newDate = new Date();
var newString = wscScripting.getDateStringFromDate(newDate);

// newString contains today's date
// e.g. 2012.01.31.16.24
```

Method: **getDirection()**
Parameters: Nil
Return Value: string containing:-
ltr = Left to Right
rtl = Right to Left e.g. Arabic
Example:

```
var direction = wscScripting.getLanguageId();

if (direction == 'rtl') {

    // This is a survey using a RTL language
}
```

Method: **getDisplayType()**
Parameters: Nil
Return Value: string containing:-
standard = Standard Display
tablet = Tablet Computer e.g. iPad
mobile = Mobile Phone / Cellular Phone e.g. iPhone
Example:

```
var display = wscScripting.getDisplayType();

if (display == 'tablet') {

    // This is a tablet based display
}
```

Method: **getDistribution()**
Parameters: Nil
Return Value: object or undefined
Example:

```
var object = wscScripting.getDistribution();

if (object) {

    // I can do something with this object
}
```

Method: **getElementById(id)**
Parameters: string - Identity of an Html Element
Return Value: object or undefined
Example:

```
var object = wscScripting.getElementById('mycontrol');
if (object) {

    // I can do something with this element
}
```

Method: **getEventData(name)**
Parameters: string - Identity of an item of data temporarily stored for later use on the current page only
Return Value: value or undefined
Example:

```
var object = wscScripting.getEventData('myvalue');
if (object) {

    // I can do something with this value
    // Value contains the text 'Hello World!'
}
```

Method: **getLanguageId()**
Parameters: Nil
Return Value: string - Two Character Language Code
Example:

```
var language = wscScripting.getLanguageId();

if (language == 'fr') {

    // This is a survey using French Language
}
```

Method: **getQuestionByDataPipingCode(dataPipingCode)**
Parameters: string - Data Piping Code of a Question - Must be a WSC Data Piping Code. If the question is not on the current page then you should use a Data Piping ShortCut to include the question on the current page
Return Value: object or undefined
Example:

```
// Using a data piping code
var object =
wscScripting.getQuestionByDataPipingCode('mydatapipingcode');
if (object) {

    // I can do something with this question
}

// Using a data piping symbol if the Question is not on the same page

// The data piping symbol with the code #data# is required to
// tell the system to have the question available
var object2 =
wscScripting.getQuestionByDataPipingCode('[@mydatapipingcode#data#@]');
if (object2) {

    // I can do something with this question
}
}
```

Method: **getQuestionByIdentity(identity)**
Parameters: string - Identity of a Question - Must be a WSC internal identity
Return Value: object or undefined
Example:

```
var object = wscScripting.getQuestionByIdentity('61ce3764-1288-
e111-8eae-0019b9c4ecf3');
if (object) {

    // I can do something with this question
}
}
```

Method: **getQuotaByCode(code)**
Parameters: string - Code of the Quota
Return Value: quota object - object of the quota
Example:

```
var oQuota = wscScripting.getQuotaByCode('GENDER');
```

Method: **getQuotaByIdentity(identity)**
Parameters: string - Identity of a Quota - Must be a WSC internal identity
Return Value: quota object - object of the quota
Example:

```
var object = wscScripting.getQuotaByIdentity('61ce3764-1288-
e111-8eae-0019b9c4ecf3');
if (object) {

    // I can do something with this quota
}
}
```

Method: **getRecallCode()**
Parameters: Nil
Return Value: string - Unique Code which identifies the response
Example:

```
var recallCode = wscScripting.getRecallCode();
```

Method: **getRowByTagValue(question, tagName, value)**
Parameters: question object - object of the question
string - name of the tag to search for
string - value of the tag being searched
Return Value: object or undefined
Example:

```
var question =  
wscScripting.getQuestionByDataPipingCode('mydatapipingcode');  
if (question) {  
  
    var row = wscScripting.getRowByTagValue(question, 'bankcode',  
    'AMER');  
    if (row) {  
  
        // I can do something with this row  
    }  
}
```

Method: **getSelectedChoices(question)**
second range
Parameters: question object - object of the question
Return Value: array or undefined
Example:

```
var question =  
wscScripting.getQuestionByDataPipingCode('mydatapipingcode');  
if (question) {  
  
    var selectedChoices =  
wscScripting.getSelectedChoices(question);  
    if (selectedChoices) {  
  
        // I can do something with this array  
    }  
}
```

Method: **getSelectedMatrixChoices(question, row)** second range
Parameters: question object - object of the question
row object - object of a row
Return Value: array or undefined
Example:

```
var question =
wscScripting.getQuestionByDataPipingCode('mydatapipingcode');
if (question) {

    var row = wscScripting.getRowByTagValue(question, 'bankcode',
'AMER');
    if (row) {

        var selectedChoices =
wscScripting.getSelectedMatrixChoices(question, row);
        if (selectedChoices) {

            // I can do something with this array
        }
    }
}
```

Method: **getSelectedRanks(question)**
Parameters: question object - object of the question
Return Value: array or undefined
Example:

```
var question =
wscScripting.getQuestionByDataPipingCode('mydatapipingcode');
if (question) {

    var ranks = wscScripting.getSelectedRanks(question);

}
```

Method: **getSubstringLeft(string, number)**
Parameters: string = Base string from which a new string will be extracted
number = Number of Characters from the Left side of the string
to be extracted
Return Value: string = Newly extracted string
Example:

```
var newString = wscScripting.getSubstringLeft('Hello World!',
5);

// newString contains 'Hello'
```

Method: **getSubstringRight(string, number)**
Parameters: string = Base string from which a new string will be extracted
number = Number of Characters from the Right side of the string
to be extracted
Return Value: string = Newly extracted string
Example:

```
var newString = wscScripting.getSubstringLeft('Hello World!',
5);

// newString contains 'orld!'
```

Method: **getTrimString(string)**
Parameters: string = Base string from which a new string will be created with spaces at either end of the string removed
Return Value: string = Newly created string
Example:

```
var newString = wscScripting.getTrimString(' Hello World!');

// newString contains 'Hello World!'
```

Method: **getValidators(question)**
Parameters: question object - object of the question
Return Value: array
Example:

```
Var question =
wscScripting.getQuestionByDataPipingCode('mydatapipingcode');
if (question) {

    var validators = wscScripting.getValidators(question);
    if (validators) {

        // I can do something with this array of validators
    }
}
```

Method: **getValue(question)**
Parameters: question object - object of the question
Return Value: value or undefined dependent on the question type
value only suitable for SingleText, MultipleText, DemographicEmail, DemographicPhone, Number, Slider and DateTime Questions
Example:

```
Var question =
wscScripting.getQuestionByDataPipingCode('mydatapipingcode');
if (question) {

    var value = wscScripting.getValue(question);
}
```

Method: **hideElement(string)**
Parameters: string = id of an Html control to hide
Return Value: boolean - confirmation that the control was hidden
Example:

```
wscScripting.hideElement('mydiv');
```

Method: **isAnyChoiceSelected(question, choices)** **second range**
Parameters: question object - object of the question
array - array of choice objects to check
Return Value: boolean - confirmation that the choice is selected
Example:

```
var question =
wscScripting.getQuestionByDataPipingCode('mydatapipingcode');
if (question && question.choices) {
    // Make an array of just 1 choice

    var arrayChoices = new Array();

    arrayChoices.push(question.choices[0]);

    var isSelected = wscScripting.isAnyChoiceSelected(question,
arrayChoices);
}
```

Method: **isAnyChoiceSelectedByValue(question, values)** **second range**
Parameters: question object - object of the question
array - array of number values to check
Return Value: boolean - confirmation that the choice is selected
Example:

```
var question =
wscScripting.getQuestionByDataPipingCode('mydatapipingcode');
if (question && question.choices) {
    // Make an array of values

    var arrayChoices = new Array(1, 2, 3);

    var isSelected =
wscScripting.isAnyChoiceSelectedByValue(question, arrayChoices);
}
```

Method: **isAnyMatrixChoiceSelected(question, choices, row)** second range

Parameters: question object - object of the question
array - array of choice objects to check
row object - object of the row

Return Value: boolean - confirmation that the choice is selected

Example:

```

var question =
wscScripting.getQuestionByDataPipingCode('mydatapipingcode');
if (question && question.choices) {
    var row = wscScripting.getRowByTagValue(question, 'bankcode',
'AMER');
    if (row) {

        // Make an array of just 1 choice

        var arrayChoices = new Array();

        arrayChoices.push(question.choices[0]);

        var isSelected =
wscScripting.isAnyMatrixChoiceSelected(question, arrayChoices, row);

    }
}

```

Method: **isAnyMatrixChoiceSelectedByValue(question, values, row)** second range

Parameters: question object - object of the question
array - array of number values to check
row object - object of the row

Return Value: boolean - confirmation that the choice is selected

Example:

```

var question =
wscScripting.getQuestionByDataPipingCode('mydatapipingcode');
if (question && question.choices) {
    var row = wscScripting.getRowByTagValue(question, 'bankcode',
'AMER');
    if (row) {

        // Make an array of values

        var arrayChoices = new Array(1, 2, 3);

        var isSelected =
wscScripting.isAnyMatrixChoiceSelectedByValue(question,arrayChoices,
row);

    }
}

```

Method: **isChoiceSelected(question, choice)** **second range**
Parameters: question object - object of the question
choice object - object of a choice
Return Value: boolean - confirmation that the choice is selected
Example:

```

var question =
wscScripting.getQuestionByDataPipingCode('mydatapipingcode');
if (question) {

    var choice = wscScripting.getChoiceByValue(question, 1);
    if (choice) {

        var isSelected =
wscScripting.isChoiceSelected(question, choice);

    }
}

```

Method: **isChoiceSelectedByValue(question, value)** **second range**
Parameters: question object - object of the question
number - value of the question choice to check
Return Value: boolean - confirmation that the choice is selected
Example:

```

var question =
wscScripting.getQuestionByDataPipingCode('mydatapipingcode');
if (question) {

    var isSelected =
wscScripting.isChoiceSelectedByValue(question, 1);
}

```

Method: **isMatrixChoiceSelected(question, choice, row)** **second range**
Parameters: question object - object of the question
choice object - object of a choice
row object - object of the row
Return Value: boolean - confirmation that the choice is selected
Example:

```

var question =
wscScripting.getQuestionByDataPipingCode('mydatapipingcode');
if (question) {

    var choice = wscScripting.getChoiceByValue(question, 1);
    if (choice) {

        var row = wscScripting.getRowByTagValue(question,
'bankcode', 'AMER');
        if (row) {

            var isSelected =
wscScripting.isMatrixChoiceSelected(question, choice, row);
        }
    }
}

```

Method: **isMatrixChoiceSelectedByValue(question, value, row)** **second range**

Parameters: question object - object of the question
number - value of the question choice to check
row object - object of the row

Return Value: boolean - confirmation that the choice is selected

Example:

```

var question =
wscScripting.getQuestionByDataPipingCode('mydatapipingcode');
if (question) {

    var row = wscScripting.getRowByTagValue(question, 'bankcode',
'AMER');
    if (row) {

        var isSelected =
wscScripting.isMatrixChoiceSelectedByValue(question, 1, row);
    }
}

```

Method: **selectChoice(question, choice)** **second range**

Parameters: question object - object of the question
choice object - object of a choice

Return Value: boolean - confirmation that the choice was selected

Example:

```

var question =
wscScripting.getQuestionByDataPipingCode('mydatapipingcode');
if (question) {
    var choice = wscScripting.getChoiceByValue(question, 1);
    if (choice) {

        var isSelected = wscScripting.selectChoice(question,
choice);
    }
}

```

Method: **selectChoiceByValue(question, number)** **second range**

Parameters: question object - object of the question
number - value of the question choice to check

Return Value: boolean - confirmation that the choice was selected

Example:

```

var question =
wscScripting.getQuestionByDataPipingCode('mydatapipingcode');
if (question) {

    var isSelected = wscScripting.selectChoiceByValue(question,
1);
}

```

Method: **selectMatrixChoice(question, choice, row)** second range
Parameters: question object - object of the question
choice object - object of a choice
row object - object of a row
Return Value: boolean - confirmation that the choice was selected

Example:

```
var question =  
wscScripting.getQuestionByDataPipingCode('mydatapipingcode');  
if (question) {  
    var row = wscScripting.getRowByTagValue(question, 'bankcode',  
    'AMER');  
    var choice = wscScripting.getChoiceByValue(question, 1);  
    if (row && choice) {  
  
        var isSelected =  
wscScripting.selectMatrixChoice(question, choice, row);  
    }  
}
```

Method: **setEventData(name, value)**
Parameters: string - Identity of an item of data temporarily stored for
later use on the current page only
object - Value of an item of data temporarily stored for later
use on the current page only
Return Value: boolean - confirmation that the value was correctly added
Example: `wscScripting.setEventData('myvalue', 'Hello World!');`

Method: **setValidation(question, text)**
Parameters: question object - object of the question
string - text of the validation message
Return Value: Nil

Example:

```
var question =  
wscScripting.getQuestionByDataPipingCode('mydatapipingcode');  
if (question) {  
    wscScripting.setValidation(question, 'Something doesnt make  
sense!');  
}
```

Method: **setValue(question, value)**
Parameters: question object - object of the question
value - type dependent on question type
value only suitable for SingleText, MultipleText,
DemographicEmail, DemographicPhone, Number, Slider and DateTime
Questions
Return Value: Nil

Example:

```
var question =  
wscScripting.getQuestionByDataPipingCode('mydatapipingcode');  
if (question) {  
  
    wscScripting.setValue(question, 'Hello World!');  
  
}
```

Method: **showElement(string)**
Parameters: string = id of an Html control to show
Return Value: boolean - confirmation that the control was shown
Example: `wscScripting.showElement('mydiv');`

Additional Objects

The following objects exist and have the properties as described with *type* and *name*.

Note: You do not have the ability to affect the rendering of a standard question by altering a property.

SurveyQuestion

string **addressType**

string **allRankedText**

array [surveychoice] **choices**

array [surveychoice] **choices2**

string **clearText**

string **containerName**

string **dataPipingCode**

number **defaultValue**

string **fieldWidth1**

string **fieldWidth2**

string **formatType**

string **gridHeadingFormat**

number **gridTotal**

string **identity**

number **increment**

number **interval**

boolean **isCommentsEnabledByDefault**

boolean **isHeadingTextVertical**

boolean **isLargeComments**

boolean **isLength**

boolean **isMandatory**

boolean **isPivot**

boolean **isQuestionOnPage**

boolean **isResetAllowed**

boolean **isSpecify**

string **javascriptBodyName**

string **listDirection**

string **listType**

number **maxIncrement**

number **maxValue**

number **minValue**

string **noneRankedText**

number **numberGrids**

string **popupType**

string **primaryRangeTitle**

string **questionNumber**

string **rankedText**

number **repeatRows**

string **resetText**

string **rowHeight1**

string **rowHeight2**

array [surveyrow] **rows**

number **scaleIncrement**

string **secondaryRangeTitle**

string **text**

string **textPosition**

string **type**

string **unRankedText**

SurveyChoice

number **grid**

string **identity**

string **imageHeight**

string **imageToolTip**

string **imageUrl**

string **imageWidth**

boolean **isComments**

boolean **isDefault**

boolean **isExclusive**

boolean **isPegged**

string **labelText**

string **numberPostText**

string **numberPreText**

array [surveychoicetag] **tags**

string **text**

number **value**

SurveyChoiceTag

string **identity**

string **name**

string **text**

SurveyRow

string **identity**

string **imageHeight**

string **imageToolTip**

string **imageUrl**

string **imageWidth**

array [surveyrowtag] **tags**

string **text**

SurveyHierarchicalListItem

string **identity**

string **description**

string **parent Identity**

SurveyRowTag

string **identity**

string **name**

string **text**

SurveyQuota

string **code**

string **identity**

bool **isPriority**

int **numberLimit**

int **numberAllowed**

int **numberResponded**

int **numberOverflow**

string **title**

SurveyDistribution

string **identity**

array [surveydistributiontag] **tags**

string **title**

SurveyDistributionTag

string **identity**

string **name**

string **text**

Browser

string **browser**

string **OS**

string **version**