

OASYS v3.5.0

Scripting

Author: Eric J. FRANCOIS

August 4, 2025

1 description

Scripting in OASYS is used predominantly to implement conditional branching in an otherwise linear test. It can read variables and act according to conditions linked to these variables.

It can also reset variables, for instance when a test is used for learning purposes and you want to loop back before an example item that was answered incorrectly.

Some arithmetic operations are also available, such as calculating sums and averages.

Finally, some internal variables of the test and the platform can be read and acted upon, such as the type of device running the test and more ...

Scripts cannot, however, read the score of the test. This limitation is currently in place to avoid the risk of cheating by reading the score in the debugger of the browser and acting upon it. This may be subject to change in the future, but for now, scripts can only react directly to the value of given answers, not to the score attributed to them.

2 context

There are 4 different contexts in which scripts can be used:

- **when navigating to a page** – this is used mainly to check if this page needs to be skipped in case of conditional branching, for instance if the questions only apply to those people who answered in a specific way to a previous question.
- **on input** – this script is executed when the user inputs an answer, for instance to set a variable based on the input.
- **when leaving a page** – on leaving a page, this script is executed, for instance to check if the user is sent back if they answered the example question incorrectly. It can also be used for more complex branching than just skipping the next page.
- **visibility conditions** – this script executes both on navigating to a page and on input, and is used to determine if a given element should be visible or not. This is a lighter way to implement conditional visibility, as it is integrated directly in the content editor and does not require a separate script to be written.

3 variables

3.1 variable types

There are 3 different types of variables in OASYS:

- **internal variables** – these are variables that are set by the platform, such as the device type, selected options for the test and overrides for the current test taker, etc. They can be read but not set.
- **local variables** – these are answers given by the test taker on the current page, such as the answer to a question or the value of a slider. They can be read and set, but only on the current page.
- **global variables** – these are answers given by the test taker anywhere in the test, such as the answer to a question on a previous page. In order to make them available throughout the test, they need to be exported to a unique variable name, which is then used to read and set the value of the variable. Global variables can be read and set anywhere in the test.

When reading a variable, local variables start with two dollar signs: `$$variable_name`; while global variables start with a single dollar sign `$variable_name`. Internal variables start with a dollar followed by two underscores `$__variable_name`.

3.2 exporting global variables

In order to export a local variable to a global variable, go to the properties of an interaction, open the **Scripting** section and enter the desired export name into the export field.

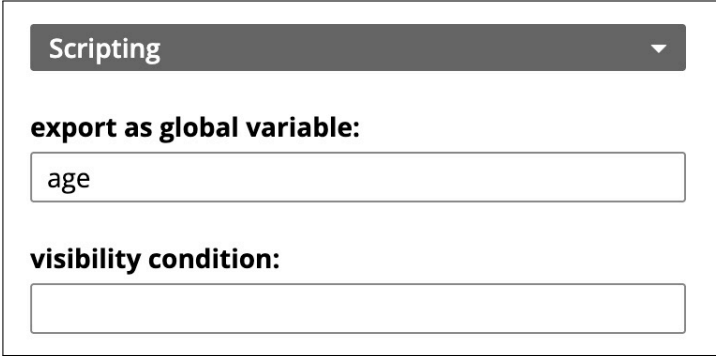


Figure 1: Exporting an interaction to a global variable

In the advanced editor, this is done by adding a `EXPORT="variable_name"` attribute to the code of the field. In case of a radio button group or a checkbox group, it's sufficient to add the attribute only in the first field of the group.

```
[@RB GROUP="gender" VALUE="female" EXPORT="gender"] [@LB]female[@/LB]
[@RB VALUE="male"] [@LB]male[@/LB]
[@RB VALUE="nonbinary"] [@LB]non binary[@/LB]
[@RB VALUE="other"] [@LB]other[@/LB]
```

Figure 2: Exporting an interaction to a global variable in the advanced editor

The export name must be unique, and it is recommended to use a descriptive name that indicates the purpose of the variable. Since pages are edited outside of the scope of a test, OASYS is not able to check for uniqueness of the variable name. This responsibility lies with the test author.

Beware: Meta fields like choice matrix or inline fields, where a single interaction creates multiple local variables, cannot be exported to global variables at the moment.

4 creating a script

4.1 creating a script in the test manager

In the test manager create a structure and then click on the script button of the page you want to create a script for.

#	Test page	Code	Page group	Label	Points	Overrides	
1	01		branching02	Regular Exercise	0	as test	</>
2	02		branching02	Regular Exercise	0	as test	</>
3	03		branching02	Regular Exercise	0	as test	</>

Figure 3: Creating a script for a page

Then you can choose between three tabs **Pre**, **Post**, **on Activity** to define scripts for different points in time. The "Pre" script will be evaluated when you are navigating to this page, the "Post" one will be run when navigating away from the page and "on Activity" will be run when an answer is given or edited.

Edit scripts for: "02"

Pre
Post
on Activity

```
if ($skipIt=="true") {skip()}
```

Figure 4: Choosing the context of a script

4.2 creating a visibility condition in an interaction

In the interaction editor, you do not have to create a complete script, but only a condition that will be evaluated to determine if the block is visible or not.

Scripting

export as global variable:

visibility condition:

Figure 5: Creating a visibility condition in an interaction

5 syntax of a script

5.1 structure of a command

Currently, 3 base commands are available in OASYS scripts:

- **IF/ELSE** – this command is used to check a condition and execute a block of code if the condition is true and execute another block of code if the condition is false.
- **IF** – this command is used to check a condition and execute a block of code if the condition is true. If the condition is false, the script will continue to the next command without executing any code.
- **EXECUTE** – this command is used to execute a block of code without conditions.

The structures of those commands are as follows:

```
if ( CONDITION ) { ACTION } else { ACTION }  
if ( CONDITION ) { ACTION }  
execute { ACTION }
```

5.2 syntax of a condition

The main syntax of a condition is as follows:

TERM1 **OPERATOR** **TERM2**

The terms can be either a global or local variable as shown previously, or they can be strings or numbers.

<code>\$\$variable</code>	local variable, e.g. the answer to a question on the current page, where the variable name corresponds to the field id; they always start with two dollar signs
<code>\$variable</code>	global variable, e.g. the answer to a question on a previous page, where the variable name corresponds to the export name of the field; they always start with a single dollar sign
<code>\$_variable</code>	internal variable, e.g. the device type, where the variable name corresponds to the name of the internal variable; they always start with a dollar sign followed by two underscores
<code>"string" or 'string'</code>	a string, written either with double or single quotes, e.g. "hello world" or 'hello world' (please make sure to use straight quotes, not curly ones)
<code>42</code>	a number, which can be either an integer or a float, e.g. 42 or 3.14 (please make sure to use a dot as decimal symbol, not a comma)

Multiple conditions can be combined using the logical operators `&&` and `||` for "and" and "or" respectively.

Example

Checking if age is between 13 and 17:

```
if ($$age >= 13 && $$age <= 17) {...}
```

5.3 syntax of an operator

The operators are used to compare the terms in a condition. The following operators are available:

<code>==</code>	equal to (for numbers or strings)
<code>!=</code>	not equal to (for numbers or strings)
<code><</code>	less than (only for numbers)
<code><=</code>	less than or equal to (only for numbers)
<code>></code>	greater than (only for numbers)
<code>>=</code>	greater than or equal to (only for numbers)
<code>contains</code>	true if a group contains a specific value (only for array-type answers like checkboxes)
<code>!contains</code>	true if a group does not contain a specific value (only for array-type answers like checkboxes)

5.4 defined actions

A set number of actions are available in OASYS scripts. They can be used in the `execute` command or in the `if` and `else` blocks.

Some commands need parameters, which are given within parentheses after the command name, separated by commas. Beware that commands which do not take parameters still need the parentheses, but they are empty.

<code>next()</code>	Navigate to the next page, just as if the "next" button had been clicked.
<code>skip()</code>	Skip the current page and proceed immediately to the next page in the test; the difference to <code>next</code> is that this command works also when navigating backwards through the test ...it then skips the current page and goes to the previous page in the test. This is only useful in the <code>pre</code> context, as it is executed before the page is rendered.
<code>goto("page_code")</code>	Navigate to the page with the given code (must be given as a string, even if numeric), just as if the user had clicked on a link to that page. The page code can be entered when creating a page, and it can be edited in the rename dialog as well.
<code>reset("page_code")</code>	Reset all answers given in the page with the given code. This is useful when creating training material and example questions are answered incorrectly.
<code>init(var, "value")</code>	Initialize a variable with a given value. The variable is always a global variable and only its name is written without the dollar sign. The value can be either a string or a number. If the variable is already set, it will be overwritten.
<code>set(var, "value")</code>	Set a variable to a given value. The variable is always a global variable and only its name is written without the dollar sign. The value can be either a string or a number. The current value of the variable will be overwritten.
<code>increment(var)</code>	Increment a variable by 1. The variable is always a global variable and only its name is written without the dollar sign. If the variable is not set, it will be initialized to 1.
<code>disableNavigation()</code>	Disable the navigation buttons on the current page. This is useful when you use branching and have multiple ending screens. Once the ending screen is reached, the user should not be able to navigate to the next page, as it belongs to a different branch of the test. Given this use case, there is no command to reenable the navigation buttons again.

<code>show("selector")</code>	Show an element on the page, where the selector is a CSS selector that identifies the element to be shown. This is a more detailed way to show an element than the visibility condition, as that one always targets the block it belongs to as a whole, while this command can target any element on the page.
<code>hide("selector")</code>	Hide an element on the page, where the selector is a CSS selector that identifies the element to be hidden. This is the exact opposite of the <code>show</code> command.
<code>avg(\$var1, \$var2, ...)</code>	Calculate the average of the given variables. This needs to be used together with the <code>set</code> or <code>init</code> command to store the result in a global variable. The input variables can be either global or local variables, and they must be numbers.
<code>sum(\$var1, \$var2, ...)</code>	Calculate the sum of the given variables. This needs to be used together with the <code>set</code> or <code>init</code> command to store the result in a global variable. The input variables can be either global or local variables, and they must be numbers.

Example

Initializing a variable with a value:

```
init(my_var, 41)
```

Incrementing a variable:

```
increment(my_var)
```

Calculating the average of two variables:

```
execute{set(my_average, avg($salary1, $salary2, $salary3))}
```

6 defined internal variables

The following internal variables are available in OASYS scripts. They can not be used in the `init`, `set` or `increment` commands, as they are read-only variables (in programming those are referred to as constants).

<code>\$_os</code>	the operating system of the device running the test, e.g. "Windows", "macOS", "Linux", etc.
<code>\$_mobile</code>	true if the test is run on a mobile device, false otherwise; this is useful to adapt the layout of the test to the device type, e.g. by hiding elements that are not needed on mobile devices
<code>\$_orientation</code>	the orientation of the device running the test, e.g. "portrait" or "landscape"; this is useful to let the test taker know they should rotate their device to the correct orientation, if that's important
<code>\$_options.limitNavigation</code>	true if the test is configured to limit navigation: test taker can only go forward, and even this is only allowed once all mandatory fields of the current page are filled in
<code>\$_options.saveResults</code>	true if the test is configured to save results, false if test is set to keep results only in memory (e.g. for example tests)
<code>\$_options.showScore</code>	true if the test is configured to show the score screen at the end of the test
<code>\$_options.useTimer</code>	true if the test is configured to use a timer, false otherwise; this is useful to know if a button to end the test should be shown at the end or not
<code>\$_options.timeLimit</code>	the time limit of the test in seconds, if the test is configured to use a timer; this shows 0 if the test is not using a timer

<code>\$_overrides.allowNavigation</code>	true if the test taker is allowed to navigate freely through the test, in spite of the test configuration; usually those test takers are beta testers, correctors or translators who might be shown extra information
<code>\$_overrides.demoMode</code>	true if the test is in demo mode, so not only results are not shown, but results in memory are erased on navigating to another page; this is used for showcasing a test in a hands on event
<code>\$_overrides.disableSaving</code>	true if the test taker is configured to not save results, in spite of the test configuration; this is used for beta testers, correctors or translators who may try out a lot and reuse the same login many times
<code>\$_overrides.disableTimer</code>	true if the test taker is configured to not use a timer, in spite of the test configuration; this may be the case for test takers with a disability who must not have a time limit; you may want to show them a button to end the test at the last page, since they will never get a time out
<code>\$_overrides.additionalTime</code>	the additional time given to the test taker, in seconds; this is useful to know if the test taker has been given extra time for a disability or other reason; this is 0 if no additional time has been given

In all likelihood, tests will have no need for most of these constants, with the exception of the orientation and the disable timer override, which have real life applications. The others are implemented for completeness and to meet needs that may arise in the future.

Beware: The information about the os is very fickle, as it is based on information given by the browser, which is known to be unreliable (e.g. Apple's handheld devices most likely claim to be a MacOS). It is not recommended to use this information in a test, as on top of companies reporting incorrect information by default, the test taker can also spoof this.

Also the `$_mobile` variable is not guaranteed, as it's based on whether a device supports multiple touch points, which may not be a good enough factor with future devices that may support multiple touch points but not be mobile devices.