APPENDIX

for

Measuring Attention in Qualtrics: Method and Validation

Table of Contents **pg.**

The purpose of this appendix is to provide QMT assistance to two segments of readers. For users seeking a ready-to-use solution with no coding involved, we provide a downloadable file and accompanying quick-start guide. For users seeking a more technical description of how QMT operates within Qualtrics, who intend to add custom features through coding, we provide a technical guide. The technical portion describes how QMT functions in Qualtrics, describes various pieces of key functionality, and provides a step-by-step guide to building custom QMT from scratch. All survey templates and code referenced in this appendix are available for use and download at: https://github.com/QMT-code/QMT.

Quick-Start Guide: The Ready-to-Use QMT Survey

We provide a ready-to-use survey that can be easily customized and modified. The mouse-tracking portion of this survey is entirely controlled using embedded data variables, and no additional programming is necessary.

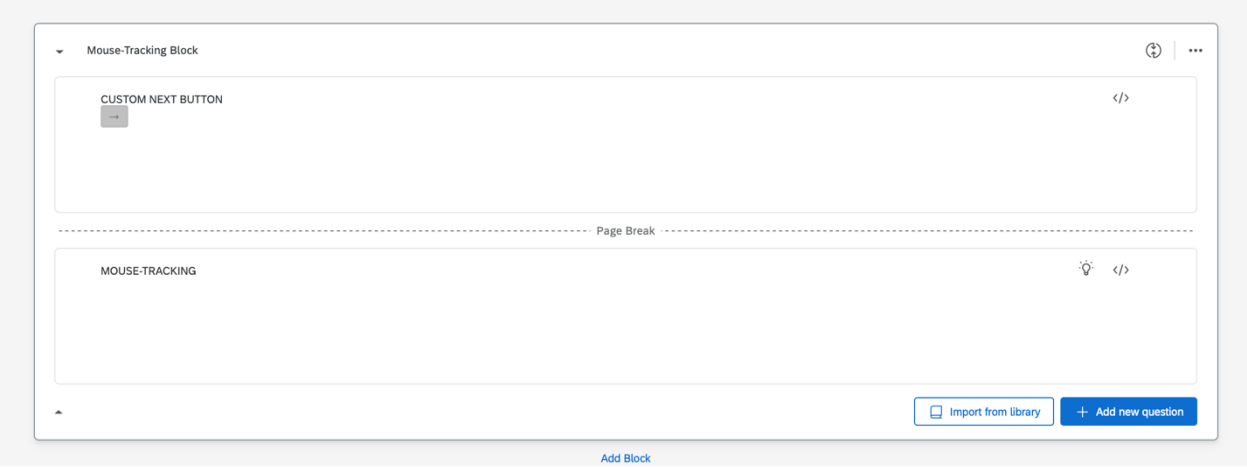
Downloading the Mouse-Tracking Survey

The ready-to-use mouse-tracking survey is available for download at: https://github.com/QMT-code/QMT. After importing the .qsf file into your library, you should be ready to preview the core mouse-tracking functionality. Of course, researchers will need to modify the ready-to-use survey to reflect their specific study design.

All Functionality Contained Within two Qualtrics Questions

The ready-to-use QMT survey contains only two Qualtrics questions: one is for mouse-tracking and the other operationalizes a custom next button (Figure W1). The question labelled "MOUSE-TRACKING" contains all mouse-tracking code, and the question labelled "CUSTOM NEXT BUTTON" overrides the Qualtrics default next button. The purpose of disabling the default Qualtrics settings is to position the next button—prior to the mouse-tracking page—in a location that does not generate unintended mouse-hovers or systematically favor certain ROIs over others. For example, if a participant advanced to the mouse-tracking screen by clicking the next button on the prior page, and if doing so positioned the participant's cursor in the exact location of a ROI, then the researcher may be incorrect to infer dwells reflect attention (as opposed to a mechanical artifact of the design). We encourage researchers to use our custom next button, for which they can set the desired page location (as will be discussed shortly).

Figure W1: The Two Code-Containing Questions for QMT.



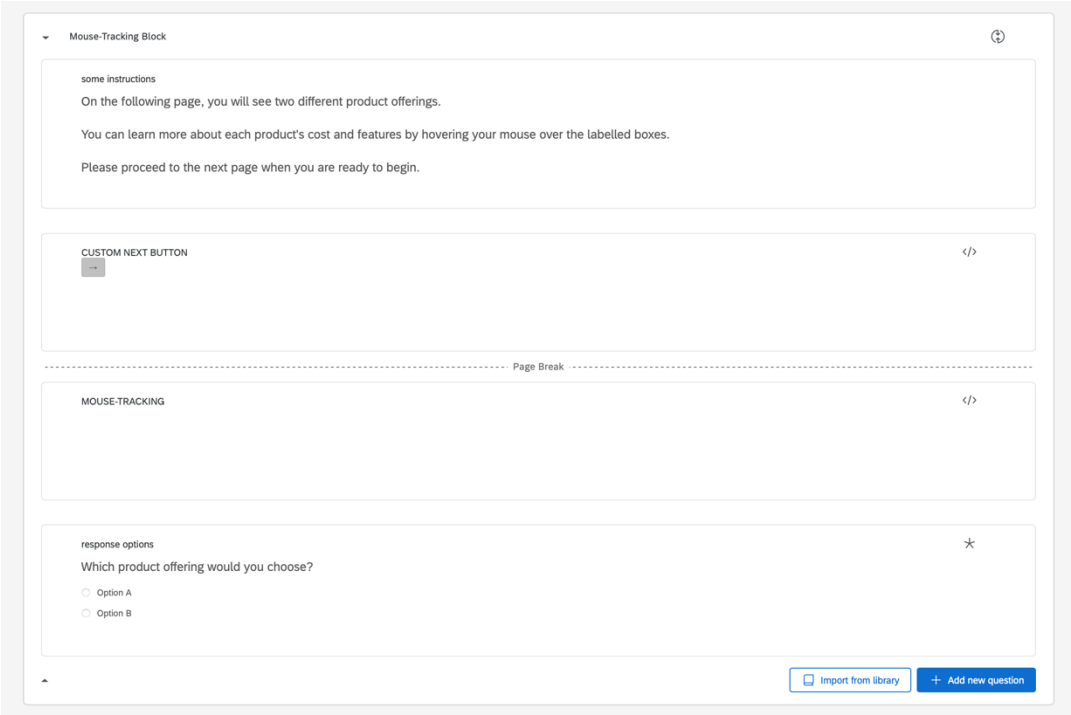
Notes: These questions can be moved, copied, modified, and combined with other questions. The question labelled CUSTOM NEXT BUTTON inserts a next button at your specified location. The question labelled MOUSE-TRACKING contains all mouse-tracking code.

In this tutorial, we differentiate between survey design and QMT customization. Survey design refers to the set of additional Qualtrics questions and non-mouse-tracking content (e.g., instructions, additional stimuli, response options, etc.), while QMT customization refers to the layout, operation, and contents of mouse-tracked questions.

Customizing the Survey Design

Survey design involves everything beyond the core functionality of mouse-tracking questions. Surveys can be constructed as normal, though special care must be taken to preserve the functionality of the two QMT questions. We recommend leaving the two QMT questions untouched and creating new questions on the same page (Figure W*2*). Advanced techniques modifying the contents of the QMT questions are discussed in the technical portion of this appendix.

Figure W2: Example Survey Design with Added Questions.



Notes: The example survey design leaves the custom next button and mouse-tracking questions untouched and adds additional questions on the same page to incorporate instructions (top question) and response options (bottom question).

The method of adding separate questions is preferred because it preserves the core functionality of the QMT questions. Any attempt to modify these questions will likely overwrite key functionality, unless the edits are made using *HTML View*. We discuss editing questions in *HTML View* extensively in the technical portion of this appendix. For the purposes of this quick-start guide, we encourage users to avoid making *any* modifications to the two QMT questions. In the event one of the QMT questions is inadvertently overwritten, this modified question should be deleted and a new (original) version of the question should be imported/copied in its place.

Customizing the Mouse-Tracking

Researchers can modify most of the mouse-tracking functionality through Qualtrics' embedded data panel. The table below contains a list of all embedded data variables and their usage instructions (Table W1).

Table W1: Embedded Data Variables.

Variable Name (in Embedded Data)	**Value**	**Usage Instructions**
timeList		Do not assign a value. This variable will be updated with a list of sequential dwell times.
regionList		Do not assign a value. This variable will be updated with a list of sequential ROIs.
mainSequence		Do not assign a value. This variable will be updated with a list of sequential dwell times, indexed by ROI.
period	1	Assign a value of 1. This variable will be used to track the trial iteration, when applicable.
delayInMs	25	Recommended value of 25. This implements a slight delay (in ms) that improves the reliability of the code.
debugMode	[0, 1]	Set this variable to 1 to turn on the debugger or 0 to turn it off.
rows	[1, k]	Set the number of rows as a positive integer value
cols	[1, k]	Set the number of columns as a positive integer value
titleLeft	[0, 1]	Set this variable to 1 to display titles to left of the mouse-tracked matrix; otherwise, set it to 0.
titleTop	[0, 1]	Set this variable to 1 to display tiles on the top of the mouse-tracked matrix; otherwise, set it to 0.
titleWidth	set as integer	Width of non-mouse-tracked title area (in pixels)
titleHeight	set as integer	Height of non-mouse-tracked title area (in pixels)
titleHorizontalSpacing	set as integer	Horizontal distance between titles and matrix border (in pixels)
titleVerticalSpacing	set as integer	Vertical distance between titles and matrix border (in pixels)
p1_titleLeft	specific formatting required	Use *** to delimit the contents of each title region. There should be (rows – 1) delimiters to separate the contents. Example usage for rows = 3 is: "these *** are *** titles"
p1_titleTop	specific formatting required	Use *** to delimit the contents of each title region. There should be (cols – 1) delimiters to separate the contents. Example usage for cols = 4 is:

		"these *** are *** also *** titles"
and… pt_titleLeft	specific formatting required	for *t* in 1:trials, create a separate variable for each *t* trial (e.g., "p2_titleLeft" and "p3_titleLeft" for trials = 3)
and…pt_titleTop	specific formatting required	for *t* in 1:trials, create a separate variable for each *t* trial (e.g., "p2_titleTop" and "p3_titleTop" for trials = 3)
boxWidth	set as integer	Width of mouse-tracked ROIs (in pixels)
boxHeight	set as integer	Height of mouse-tracked ROIs (in pixels)
horizontalSpacing	set as integer	Horizontal distance between ROIs (in pixels)
verticalSpacing	set as integer	Vertical distance between ROIs (in pixels)
p1_hidden	specific formatting required	Use *** to delimit the hidden contents of each ROI. There should be (rows * cols - 1) delimiters to separate the contents.
p1_revealed	specific formatting required	Use *** to delimit the revealed (displayed by default) contents of each ROI. There should be (rows * cols - 1) delimiters to separate the contents.
and… pt_hidden	specific formatting required	for *t* in 1:trials, create a separate variable for each *t* trial (e.g., "p2_hidden" and "p3_hidden" for trials = 3)
and… pt_revealed	specific formatting required	for *t* in 1:trials, create a separate variable for each *t* trial (e.g., "p2_revealed" and "p3_revealed" for trials = 3)
buttonX	[5, 95]	Set the custom button location, based on the percent of the way to the left edge of the screen
buttonY	[5, 95]	Set the custom button location, based on the percent of the way to the bottom edge of the screen
alignHorizontal	[left, center, right]	Set the horizontal alignment (i.e., left-justified, centered, or right-justified) of your contents
alignVertical	[top, center, bottom]	Set the vertical alignment (i.e., aligned top, centered, aligned bottom) of your contents

Notes: All included embedded variables from the ready-to-use survey. The left column identifies the variable name; the center column identifies the set of values that can be assigned; and the right column provides additional usage instructions.

The variables ***timeList***, ***regionList***, and ***mainSequence*** are instantiated but not assigned any value in embedded data. The purpose of these variables is to declare where the mouse-tracking data should be stored, after it is collected. Variables *timeList* and *regionList* will contain

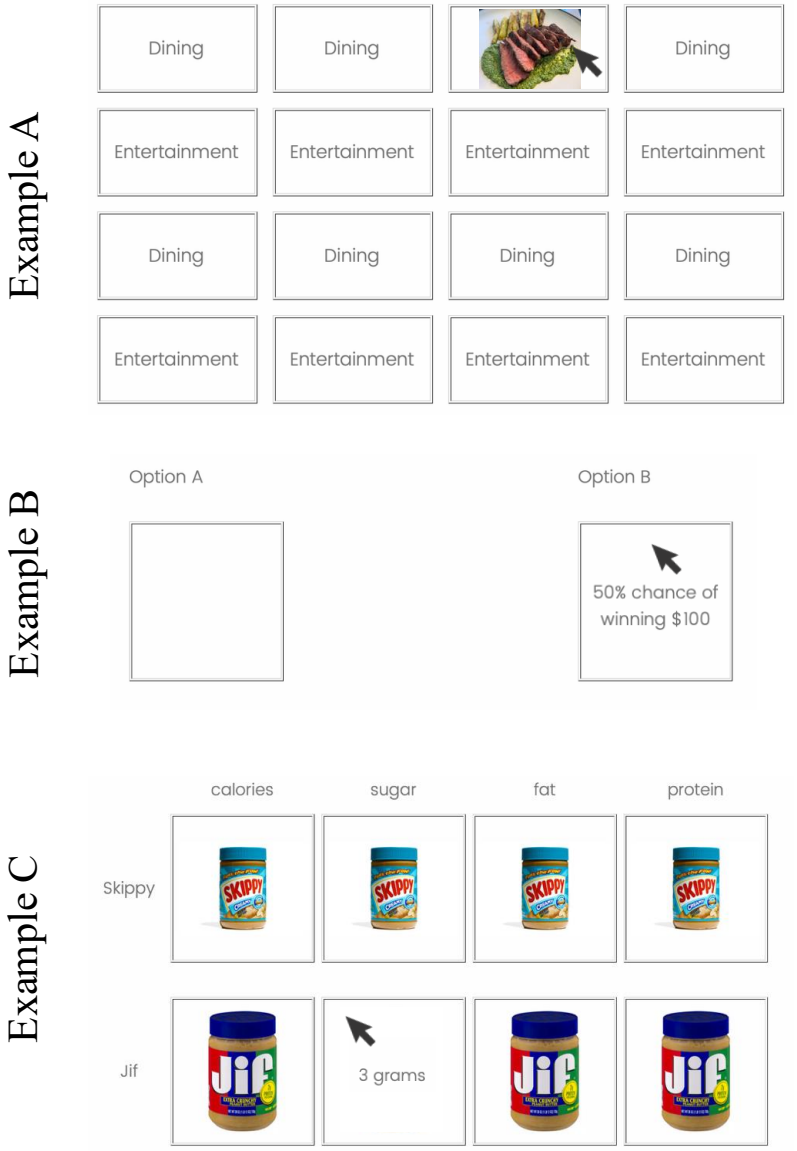
ordered lists of dwell times and ROIs visited, respectively. The variable *mainSequence* will contain an ordered list of time-ROI pairs.

Regardless of specific survey design choices, the code expects a variable called ***period***, which should be initially set to 1. This variable tracks the number of mouse-tracking periods (trials). For designs with a single instance of mouse-tracking, this variable may not appear to serve an important function (though it is necessary for the QMT code). However, in designs featuring multiple instances of QMT—like multiple different mouse-tracked questions, blocks, or repeated *Loop and Merge* questions—this variable indexes the period for mouse-tracked data. For example, if a single QMT question is structured within a *Loop and Merge* to be presented a total of three times, the period variable is used to (i) tell the mouse-tracking question which data to present on each iteration and (ii) how to store the data so that the researcher can identify which dwells occurred in a given loop. The period variable increments after each mouse-tracking question is presented, regardless of whether there are multiple copies of the mouse-tracking question, or a single mouse-tracking question that iterates through a *Loop and Merge*.

We suggest using a slight delay (i.e., ***delayInMs*** = 25), to help with the reliability of code execution, which is further discussed in the technical guide (later in Step 4).

To assist with debugging and visualizing real-time mouse-tracking data, there is a debugging tool that can be turned on (***debugMode*** = 1) or off (***debugMode*** = 0). The debugger is only for testing purposes and should always be turned off for data collection.

Figure W3: Various Layout Examples.



Notes: Examples of three different layouts of QMT. Example A is a 4 x 4 grid with centered text in each ROI (in the non-hover state). Hovers reveal images. Example B is a 1 x 2 grid with a high degree of horizontal spacing. This example also includes left-justified titles and does not provide any information in the non-hover state. Example C is a 2 x 4 grid of images and includes centered titles to the left of the grid and on top of the grid.

The layout of your QMT page begins by specifying the number of ***rows*** *and* ***columns***. These variables require a non-negative integer, and the minimum possible value is 1. Therefore, the total number of ROIs will be the product of the number of rows and number of columns. By

default, the program will regard the top-left region (row 1, column 1) as the first region, and then will work from left to right before proceeding to the next row. (An easy method to verify the numerical identity of a region is to hover over it with the debugging tool on, and to check how the mouse-tracking data are being stored.) While there is not technically a maximum number of rows or columns, page size constraints will be important to consider. Various page layout examples are provided in Figure W3.

The examples in Figure W3 highlight the different options for including non-mouse-tracked titles, which are controlled by ***titleLeft*** and ***titleTop***. When *titleLeft* is set to 1, each row displays a title (if provided) to the left of the mouse-tracked grid. When *titleTop* is set to 1, each column displays a title (if provided) above the mouse-tracked grid. (In Figure W3, Example A does not use titles, Example B uses only *titleTop*, and Example C uses both *titleLeft* and *titleTop*.)

Counterbalancing using embedded data

Titles and mouse-tracked ROIs get their content directly from embedded data. Using embedded data allows researchers to easily incorporate additional Qualtrics features, such as randomization and branch logic. For example, researchers could use a randomizer in embedded data to counterbalance the order of contents (e.g., defining contents as “A, B” and also “B, A”, and randomizing which order a participant receives).

Modifying the question-specific content

Irrespective of whether the researcher uses any form of randomization or counterbalancing, setting the content of the titles and mouse-tracked ROIs requires a precise formatting of both the variable name and variable contents. Regarding the variable naming conventions: Titles, hidden content (displayed only during a hover), and revealed content

(displayed by default, without a hover) each require an embedded data variable that indexes the intended period. Each of these variables begins with *pt_*, where *t* is the time period in which the contents should be shown. For example, *p1_*, *p2_*, and *p3_*, would all be included in the variable names for contents to be displayed during periods 1, 2, and 3, respectively. All surveys will have at least one period by default, so ***p1_titleLeft*** and ***p1_titleTop*** are used to store title contents, and ***p1_hidden*** and ***p1_revealed*** are used to store hidden and revealed contents, respectively. There should be an additional variable created for each trial in the survey design (e.g., adding in ***p2_hidden***, ***p2_revealed***, ***p3_hidden***, ***p3_revealed*** for a three-period design).

Delimiting

To set the variable contents of each of these variables, your content must be delimited by three asterisks (***). This is how the QMT code separates distinct bits of text. For example, to set the revealed content of three regions as Region 1, Region 2, and Region 3, you would define *p1_revealed* as the asterisk-delimited string: Region 1***Region 2***Region 3. This approach is used for titles and ROI contents, so the code to set top titles in period 1 would look like: *p1_titleTop* = Option A***Option B.

Images

As depicted in Figure W3, you can also insert **images** (in the form of image URLs). These can be externally hosted images (e.g., the URL of an image from a website) or internally hosted images (e.g., images you upload to your Qualtrics library).[1] In order to indicate that you will be using an image—as opposed to regular text—you must append the following text,

[1] A straightforward method to obtain the URL of an internally hosted image is to (1) upload the image—as you normally would—to a different question, and then (2) use *HTML View* on that question to copy the full URL.

without quotations, before the image's URL: "IMAGE_URL:". For example, you would insert title images as: *p1_titleTop* = IMAGE_URL:https://whatever_your_url_is*** IMAGE_URL:https://whatever_your_other_url_is.

Dimensions

In addition to setting the contents (of both non-mouse-tracked titles and the mouse-tracked ROIs), users can also specify certain formatting aspects of the content.[2] The size of the non-mouse-tracked title area (if applicable) is set by ***titleWidth*** and ***titleTop***, both of which should take a positive integer value to specify the size, in pixels. The variable ***titleHorizontalSpacing*** determines the distance between the *titleLeft* and the left-most edge of the mouse-tracking grid. The variable ***titleVerticalSpacing*** determines the distance between the *titleTop* and the top of the mouse-tracking grid. Relatedly, the size of the mouse-tracked ROIs is controlled by ***boxWidth*** and ***boxHeight*** (also in pixels). The space between rows and columns is set by ***verticalSpacing*** and ***horizontalSpacing***, respectively.

Alignment

You can also set the alignment of your contents within the title areas and ROI dimensions you determine. The variable ***alignHorizontal*** allows you to specify left-, center-, or right-alignment using keywords: left, center, right. The variable ***alignVertical*** allows you to specify top-, center-, or bottom-alignment using keywords: top, center, bottom.[3]

[2] Images are automatically formatted to their maximum dimensions—maintaining original scale—based on the boxWidth and boxHeight parameters.
[3] See the prior footnote.

Next button

Finally, you can set the location of the custom next button through the variables ***buttonX*** and ***buttonY***. Both variables expect a value in the range [5, 95].[4] For *buttonX*, the value corresponds to the percent of the way to the left edge of the screen (moving from right to left). For *buttonY*, the value corresponds to the percent of the way to the bottom of the screen (moving from top to bottom). Therefore, *buttonX* = 5 and *buttonY* = 5 corresponds to the top-right-most corner; *buttonX* = 50 and *buttonY* = 50 corresponds to the center of the page; and *buttonX* = 95 and *buttonY* = 95 corresponds to the bottom-left-most corner. As previously discussed, we suggest you place the custom next button on the page *preceding* the mouse-tracking (and set the location such that the default cursor location does not overlap any ROI).

At this point, we invite readers to test the ready-to-use QMT survey and experiment with the embedded data controls. To assist in data analysis, we provide additional R code to help unpack the QMT mouse-tracking data, which is available for download through the GitHub page and also discussed later in this appendix (Step 11).

[4] We do not recommend values less than 5 or greater than 95 for risk of extending the next button to non-visible regions of the page.

Technical Overview: Workflow and Data Storage for QMT

The preceding quick-start guide is intended to make Qualtrics Mouse-Tracking accessible to users without the need to modify any code. Some researchers (and some survey designs) may benefit from making such coding modifications. This section is for anyone interested in a more technical description of how custom code interacts with Qualtrics. In this technical guide, we propose a set of best-practices—complete with code snippets—to build mouse-tracking features into any Qualtrics survey.

Background on QMT

Qualtrics mouse-tracking involves inserting custom HTML, JavaScript, and CSS code into the Qualtrics environment. HTML (HyperText MarkUp Language) is used to define and organize a webpage, and is structured through a set of elements contained within tags (e.g., and are the opening and closing tags that wrap around text to make a bolded element). The custom mouse-tracking HTML communicates with JavaScript, which executes code to interact with the client's browser. The JavaScript is responsible for tracking the user's behavior on the webpage, which HTML renders. More generally, JavaScript is the portion of code that implements changes to the webpage's data, conducts calculations, and updates and saves data. CSS (Cascading Style Sheets) is a style sheet language that modifies the appearance of HTML elements. We use this to update the appearance of regions of interest and the custom next button. Therefore, the collective purpose of the HTML, JavaScript, and CSS is to incorporate new mouse-tracking features and to overwrite certain default settings.

The general technique recommended in this guide is to leverage HTML event attributes to track when a user's mouse enters certain predefined regions within a Qualtrics question. HTML events are browser-initiated events that occur when the user interacting with the webpage does something (e.g., clicks a button, scrolls, or moves their mouse). Once the webpage registers an HTML event—in this case, a mouse entering or leaving a predefined region of interest—it communicates with custom JavaScript code to keep track of this behavior. The JavaScript code then collects time stamps to track the dwell time within a given region, until another HTML event signals the user has moved their mouse beyond the region. JavaScript is then responsible for saving all relevant information (i.e., the region that was visited, the duration of the visit, and the sequence of other region-duration visits) in a manner that is compatible with Qualtrics' embedded data. Therefore, all data are recorded and stored within Qualtrics and are included in the standard output file.

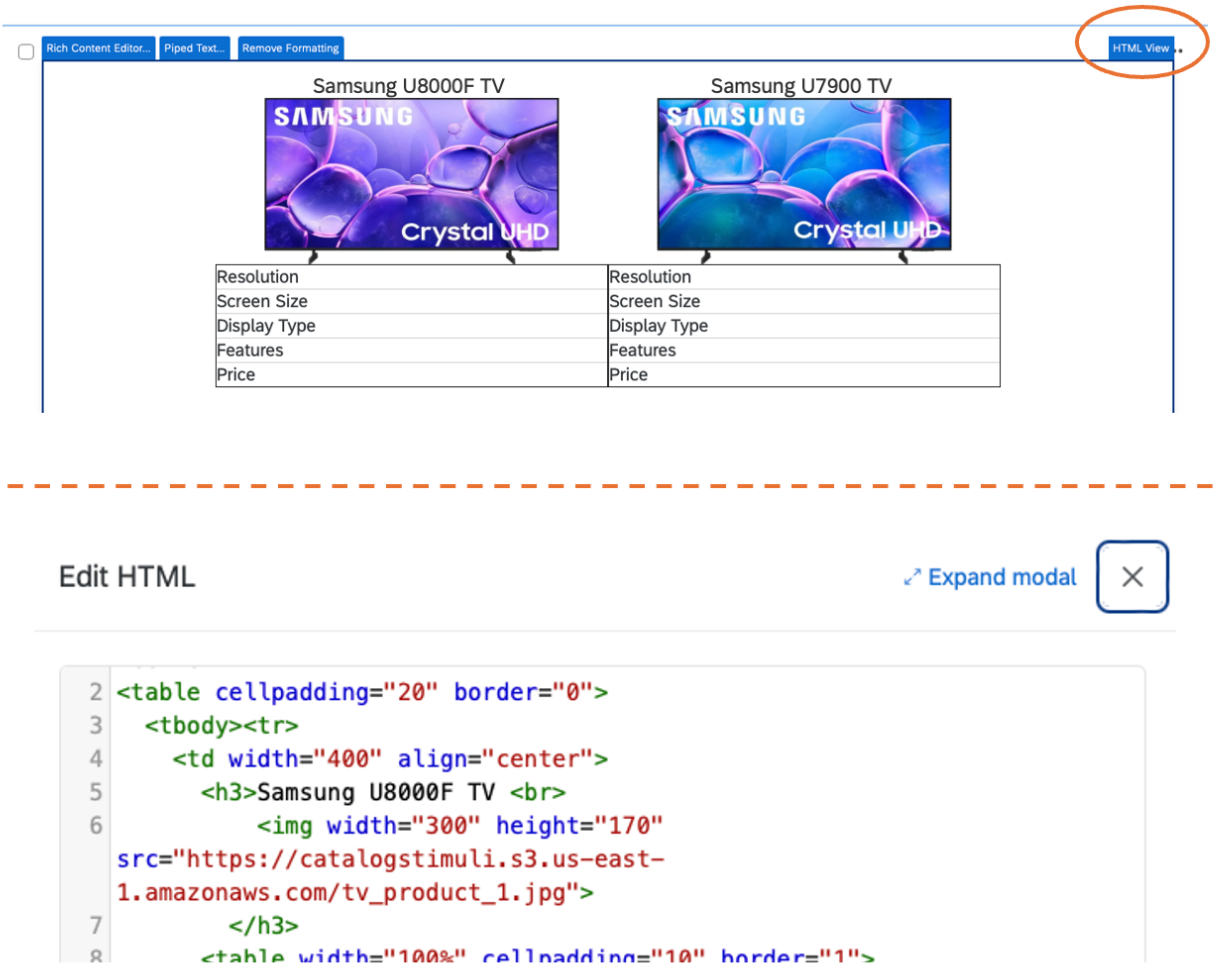
Accessing and Interacting with HTML

Qualtrics mouse-tracking involves custom snippets of HTML code to construct the mouse-tracked regions of interest. It is imperative to interact with any question containing custom mouse-tracking through Qualtrics' *HTML View* (Figure W4). You may inadvertently overwrite custom HTML if you attempt to modify a Qualtrics question through the standard approaches (i.e., clicking on the question and typing, using the *Rich Content Editor*, or using the *Piped Text* button). Overwriting HTML by using any approach other than *HTML View* can be difficult to diagnose, as some elements may be overwritten, while others remain.

As a best-practice, we recommend HTML is placed within each question's *HTML View*, as previously discussed. We note that as an advanced alternative, HTML code can also be

constructed or modified as character strings in JavaScript, which is the coding approach we take in the ready-to-use survey. In general, it is preferable to avoid this method due to the added complexity; however, an interested reader can examine the JavaScript of our ready-to-use survey for additional details.

Figure W4: Accessing the HTML View.



Notes: The *HTML View* is accessible by first clicking on the content of a Qualtrics question (e.g., the default content for a new question is "Click to write the question text") and then clicking the button located in the top right corner (upper image). Clicking the *HTML View* button will open a new window with the HTML editor (lower image).

Accessing and Interacting with JavaScript

JavaScript code can be stored and accessed using three different methods. First, JavaScript can be contained alongside a question's HTML through the *HTML View*. This method requires wrapping any JavaScript code in HTML <script> tags. Second, a JavaScript file that is stored as readable on an external server (e.g., on GitHub or another code repository) can be accessed directly using an HTML <script src = *URL*> call. The third—and preferred—method is to use Qualtrics' *Question JavaScript Viewer*, which can be accessed by clicking on the "</> *JavaScript*" button. After JavaScript has been added through this viewer, you can use the shortcut "</>" button, which will appear on the top-right of each question with existing JavaScript.

We recommend using Qualtrics' Question JavaScript Viewer because it provides the most flexibility and stability. Compared to adding JavaScript through the *HTML View* using <script> tags, you can add more overall code using the JavaScript Viewer. Whereas the *HTML View* has a 20,000-character limit (inclusive of HTML, JavaScript, and CSS), the limits when using the Question Viewer are much higher. Furthermore, the JavaScript Question Viewer includes three optional functions called *addOnload*, *addOnReady*, and *addOnUnload*, which can be helpful for structuring the timing of when custom code is executed. These functions are especially useful for interacting with the mouse-tracked content after the page is fully loaded, which is needed for certain error-catching features, as will be discussed.

Beyond the functionality of the timing functions contained within the JavaScript Viewer, we recommend placing JavaScript code here for two reasons. First, the JavaScript Viewer will

attempt to compile your code (upon saving) to check for any major coding syntax errors that might prevent the JavaScript from running. This is not the case when using HTML <script> tags in the *HTML View*. Second, whereas a user may accidentally delete or overwrite JavaScript in <script> tags (e.g., by typing without using a question's *HTML View*), the JavaScript Viewer protects code from being overwritten.

Accessing and Interacting with CSS

Similar to JavaScript, CSS may be stored within *HTML View* (using <style> tags), accessed from an external host (using <style src = *URL*> tags), or contained within the Qualtrics *Custom CSS* field. Best-practice is to use the *Custom CSS* field (contained within *Look and Feel*), for the purposes of flexibility and stability. Beyond the parallel benefits already described in the prior section on JavaScript, Qualtrics' *Custom CSS* field allows you to preview any CSS updates in real-time.

Data Storage (in Qualtrics)

All relevant user data in Qualtrics are stored as embedded data. Embedded data fields do not naturally accommodate traditional data structures (e.g., vectors, matrices, etc.); however, it is possible to store data as character strings that may be later converted to more traditional data structures. Given these constraints, we propose storing mouse-tracking data as a character string in which separate pieces of information (e.g., information at separate indices) are clearly delimited with special characters. In the provided code, we use the triple asterisk (***) as the delimiter to create distinct indices. Using this approach, the string "Mouse-tracking***is***fun" can be converted to a traditional data structure with unique indices by splitting on the delimiter

(e.g., [mouse-tracking, is, fun], where “mouse-tracking” is the content of the first index, “is” is the content of the second index, and “fun” is the content of the third index).

In addition to storing a sequence in embedded data, an additional requirement is to capture the relevant information about which regions were visited and for how long. To achieve this, best-practice is to append specific identifiers to each observation of mouse-tracking. We suggest using the form: *Px_Ry:t*, where *x* is the period counter, *y* is the region identifier, and *t* is the dwell time, in milliseconds. Using this form might result in an embedded data string such as “P1_R2:350,P1_R2:600,P1_R1:380,P1_R3:400”. This would correspond to four unique mouse-tracked events, all in the first period: visiting region 2 for 350 ms, visiting region 2 for 600 ms, visiting region 1 for 380 ms, and visiting region 3 for 400 ms. We provide a file in R to unpack these character strings and transform them into a long data structure (Step 11).

Step-by-Step Guide to Implementing Qualtrics Mouse-Tracking

This portion of the technical appendix provides a step-by-step guide to building a QMT survey from scratch. Readers following these steps will construct their own, fully functional survey, which can then be further customized.

Step 1 – Define Embedded Data

You will need to define several variables in the Qualtrics embedded data, accessed through the *Survey Flow*. You can define variables by navigating to *Survey Flow*, clicking *Add a New Element Here*, selecting *Embedded Data*, and selecting *Add a New Field*. For the purposes of QMT, embedded data variables serve two purposes: data storage and parameter values.[5]

One set of variables (*timeList*, *regionList*, and *mainSequence*) are names that will eventually be used to store mouse-tracking data. You will define these variables without assigning any values. The variable *timeList* will store the sequence of dwell times; the variable *regionList* will store the sequence of regions visited; the variable *mainSequence* will store all combined region-time information. Therefore, all essential mouse-tracking data will be captured within these three embedded data variables.[6]

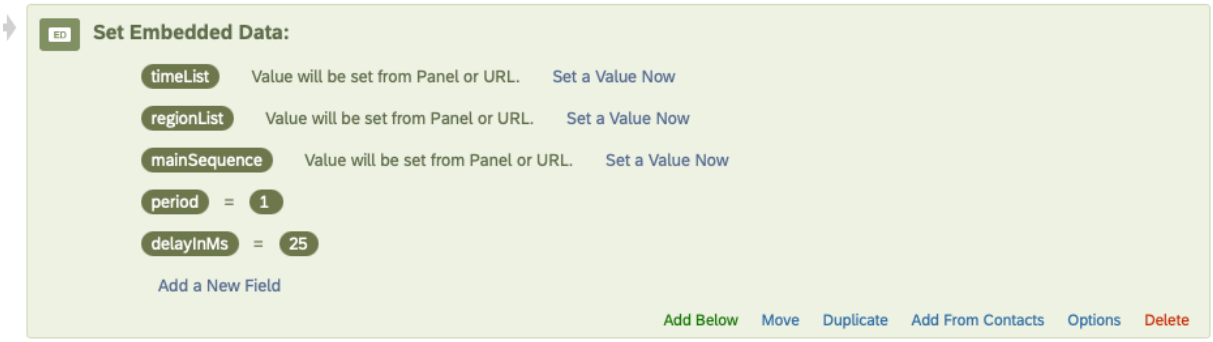
Other variables act as parameters with defined values that are passed to JavaScript. These variables are *period* and *delayInMs*. You must assign the value 1 to period (period = 1), which is used to index the period in repeated-measure environments.[7] This period variable should be

[5] Whereas the ready-to-use survey featured many embedded data variables to avoid direct access to HTML and JavaScript (see Table W1), building custom QMT surveys requires very few embedded data variables.
[6] Technically, the mainSequence alone captures all mouse-tracking data. We encourage the inclusion of timeList and regionList for readability and ease of access to data.
[7] If period is undefined in embedded data, the JavaScript will set a value of 1.

defined and assigned a value of 1, regardless of whether your survey structure represents a single period, multiple different mouse-tracked questions, or a single mouse-tracked questions repeated multiple times (e.g., contained within a Loop and Merge). Additionally, we recommend creating and assigning the variable *delayInMs* = 25.[8] You may adjust this timing parameter, which is intended as a mostly imperceptible delay before revealing a region's contents upon a mouse over. The purpose of this delay is to improve the reliability of the mouse-tracking by decreasing the likelihood that the show() and hide() functions operate out of order. Importantly, the mouse-tracking timer does not start until the hidden content is revealed, and any fleeting mouse overs that are less than the configured delayInMs will neither reveal information nor be stored in data.

In summary, Step 1 involves defining the following variables in Qualtrics embedded data: *timeList, regionList, and mainSequence*; as well as *period* = 1 and *delayInMs* = 25 (Figure W5).

Figure W5: Defining Embedded Data Variables.



Notes: These variable names should be defined in Qualtrics' *Embedded Data*. The variable *period* should be assigned a value of 1, and we recommend a value of 25 for *delayInMs*.

[8] If delayInMs is undefined in embedded data, the JavaScript will set a value of 0.

Step 2 – Create HTML Table Structure

Prior to working with any custom code snippets, you must first establish the layout for the mouse-tracked regions of interest. At minimum, this requires (i) creating a question in Qualtrics and (ii) modifying the HTML of that question to include one or more tables. Throughout this guide, we use tables as the visual framework to organize the mouse-tracked regions. We prefer to insert mouse-tracking capabilities into tables—more specifically, table cells—because they have a clearly defined spatial layout that is relatively straightforward to modify. Tables make it easy to clearly define regions of interest and place them in specific locations on the screen. You may consider using a single table with multiple ROIs or multiple tables each with a single ROI. Such flexibility is precisely what makes HTML tables appealing for this application. As a matter of best practice, we recommend using a table with a border to clearly define the boundary of your regions of interest to participants. You may add a border to all tables using custom CSS (not shown) or by including the "border" argument when you define your HTML table, within the <table> tag (i.e., <table border = "1">) as shown in Figure W6.

There are various ways to obtain HTML table code. For readers familiar with writing in HTML, the simplest approach may be to manually write the code in your specific question's *HTML View*. Alternatively, widely available online resources can produce HTML code, which can be pasted in the *HTML View*. Additionally, Qualtrics' *Rich Content Editor* graphical user interface can be used to generate an initial HTML table (however, it is imperative not to use the *Rich Content Editor* after any HTML has been modified with additional mouse-tracking code). Finally, you can modify the included code to reflect your desired table structure. The HTML below provides the basic table structure that includes two data cells of a specified size (Code W1). As in the provided code, we recommend keeping at least one plain text element (e.g., "This

is some placeholder text that occurs before your mouse-tracked table.") to prevent Qualtrics from overwriting HTML that does not include any plain text.[9]

Code W1: HTML for Basic Table Structure.

```
This is some placeholder text that occurs before your mouse-tracked table.
<table border = "1">
    <tbody>
        <tr>
          <td style="width: 300px; height: 300px;"></td>
          <td style="width: 300px; height: 300px;"></td>
        </tr>
    </tbody>
</table>
```

Notes: Using *HTML View*, paste the entire snippet of code—including the plain text about serving as a placeholder—into your question.

The HTML for your bordered table should be written (or pasted) into the specific question you intend to mouse-track. Specifically, all HTML must be placed within the question using the *HTML View*. An example of HTML for a simple, two-cell table (with a border) is provided in the upper portion of Figure W6. Note, after placing the HTML code into a Qualtrics question, you will need to use *HTML View* exclusively for all further updates to a mouse-tracked question. Using the standard question editing mode or the *Rich Content Editor* after modifying the HTML may erase or overwrite your custom code.

[9] For example, placing the empty table <table border = "1"><tbody><tr><td></td></tr></tbody></table> into *HTML View* would be inadvertently overwritten in the Qualtrics software if the question is ever clicked on. This is an undesirable Qualtrics feature and can be avoided by including at least one piece of plain text to stabilize the code.

Step 3 – Insert the Event Listeners in HTML

After generating your HTML table, you will need to modify the code with snippets to enable browser-based mouse-tracking. Best-practice is to modify the table data associated with each cell (the opening <td> tag) to include an *onmouseenter* and *onmouseleave* HTML events (Figure W6). While every mouse-tracked cell will include the same HTML event listeners (*onmouseenter* and *onmouseleave*), the values assigned to each cell's event listeners must call uniquely identifiable functions. To modify the HTML provided in Step 2, you would simply replace the two <td></td> lines with the following code that defines the data cells to include the HTML listeners (Code W2).

Code W2: HTML to add Event Listeners.

```
<td id="Region1" onmouseenter="showR1()" onmouseleave="hideR1()"></td>
<td id="Region2" onmouseenter="showR2()" onmouseleave="hideR2()"></td>
```

Notes: Using *HTML View*, paste to overwrite the empty data cells—i.e., <td></td>—of the table where you intend to add mouse-tracking features.

These functions, which will be later defined in JavaScript, are responsible for showing and hiding the mouse-tracked information. For clarity, we will generally refer to these as the show() and hide() functions; however, each HTML table data cell (<td>) will call its own unique show() and hide() functions. For example, in a design with only two mouse-tracked regions, you might have functions showR1() and hideR1() for mouse interactions with Region 1, and showR2() and hideR2() for interactions with Region 2. In addition to each mouse-tracked region having its own unique show() and hide() function names, it is also necessary to give each cell a uniquely identifiable name (e.g., *id = "Region1"* and *id = "Region2"*) (Figure W6).

Figure W6: HTML Table Structure With and Without Mouse-Tracking.

Edit HTML

```
1  This is some placeholder text that occurs before your mouse-tracked table.
2  <table border = "1">
3      <tbody>
4          <tr>
5             <td style="width: 300px; height: 300px;"></td>
6             <td style="width: 300px; height: 300px;"></td>
7          </tr>
8      </tbody>
9  </table>
10
```

```
1  This is some placeholder text that occurs before your mouse-tracked table.
2  <table border = "1">
3      <tbody>
4          <tr>
5            <td id="Region1" style="width: 300px; height: 300px;" onmouseenter="showR1()" onmouseleave="hideR1()"></td>
6            <td id="Region2" style="width: 300px; height: 300px;" onmouseenter="showR2()" onmouseleave="hideR2()"></td>
7          </tr>
8      </tbody>
9  </table>
10
```

Notes: Example view of *HTML Mode* for a bordered, two-cell table in HTML without mouse-tracking (upper) and with mouse-tracking (lower). The only differences in code occur in lines 5-6, within the <td> opening tags.

Step 4 – Insert JavaScript Before the Timing Functions

This guide provides a JavaScript template that can be modified to accommodate various mouse-tracking applications. The core functions of this code are to (1) observe the time when a mouse enters a region of interest; (2) observe the time when a mouse exits the same region of interest; and (3) store information about the duration of the dwell in each ROI. The code operates by using absolute time stamps (from the JavaScript Date.now() function) to calculate the dwell time between a mouse enter and a mouse leave, in each region. Each time a mouse leaves a region of interest, the hide() function updates Qualtrics embedded data with a record of the event. Additional failsafe code is intended to protect this functionality by (i) adding slight delays to decrease the likelihood of functions executing too quickly (i.e., the hide() function attempting

to conceal content before the show() function has finished rendering it) and (ii) safeguarding against malfunctions caused by erratic mouse movements or browser errors.

The JavaScript code snippets must be located at precise locations within the Qualtrics environment to ensure the HTML event handlers have access to the custom JavaScript functions (at the correct times). All code defining the custom show() and hide() functions should be placed before the three custom JavaScript functions named *addOnload*, *addOnReady*, and *addOnUnload* (Figure W7). This code position is critical to ensure the custom functions have a global scope and are loaded before the HTML loads and renders.

Figure W7: The JavaScript Viewer.

Notes: Most of the custom JavaScript code—including the creation of the show() and hide() functions, and their configuration objects—must be placed before the three timing functions (*addOnload*, *addOnReady*, and *addOnUnload*). Other portions of code that require interacting the HTML objects (e.g., error-checking code) will be placed in the timing functions in Steps 6-8.

For readers following along with the code examples from Steps 2 and 3, the following code can be used with the existing HTML structure (Code W3).

Code W3: Core JavaScript for QMT.

```
// Read lists from embedded data
        var timeListString = "${e://Field/timeList}";
        var regionListString = "${e://Field/regionList}";
        var mainSequenceString = "${e://Field/mainSequence}";

// Convert to arrays (just sets type to array if empty)
        var timeList = timeListString ? timeListString.split(",") : [];
        var regionList = regionListString ? regionListString.split(",") : [];
        var mainSequence = mainSequenceString ? mainSequenceString.split(",") : [];

// Global timing tracking
var regionStartTimes = {};
var showTimeouts = {};

// Delay variable
var delayInMs = parseInt("${e://Field/delayInMs}") || 0;

// This is the number that will be appended to the embedded data list contents
var period = parseInt("${e://Field/period}") || 1;

// Configuration object for all regions
/*
You should modify this code to reflect your study design.
The number of unique key-value mappings (e.g., "Region1: {name: ...}" is 1 such mapping)
should reflect the number of mouse-tracked regions in your HTML. The key (e.g., "Region1")
must match the id from each region in your HTML (e.g., "<td id = "Region1" ...). The name
value (for each key) should reflect some systematic naming convention that will allow you to
identify the region. We use "R1", "R2", ... "RK" as names of Regions 1, 2, ... K. You will need to
update your hiddenContent (hidden by default) and revealedContent (revealed by default) for
each key.
*/


var regionConfig = {
   "Region1": {
      name: "R1",
      hiddenContent: "Region 1 Test",
      revealedContent: ""
   },
   "Region2": {
```

```
        name: "R2",
        hiddenContent: "Region 2 Test",
        revealedContent: ""
    }
};

// Function to initialize the HTML with revealedContent from JavaScript
function initializeRegionContent() {
    Object.keys(regionConfig).forEach(function(regionId) {
        var element = document.getElementById(regionId);
        if (element) {
            element.textContent = regionConfig[regionId].revealedContent;
        }
    });
}

// Core functions
function showRegion(regionId, hiddenContent) {
    // Clear any pending show for this region
    if (showTimeouts[regionId]) {
        clearTimeout(showTimeouts[regionId]);
        delete showTimeouts[regionId];
    }

    // Delay show to stabilize fast mouse movements
    showTimeouts[regionId] = setTimeout(function() {
        var element = document.getElementById(regionId);
        if (!element) return;

        // Only show if not already shown
        if (!regionStartTimes[regionId]) {
            element.innerHTML = hiddenContent;
            regionStartTimes[regionId] = Date.now();
        }

        delete showTimeouts[regionId];
    }, delayInMs);
}

function hideRegion(regionId) {
    // Clear any pending show for this region
    if (showTimeouts[regionId]) {
        clearTimeout(showTimeouts[regionId]);
        delete showTimeouts[regionId];
    }
```

```
    // Hide immediately - no delay
    var element = document.getElementById(regionId);
    if (!element) return;

    // Get the revealedContent from config
    var config = regionConfig[regionId];
    element.innerHTML = config.revealedContent;

    if (regionStartTimes[regionId]) {
        // Calculate time spent
        var timeSpentMs = Date.now() - regionStartTimes[regionId];

        // Update sequences
        timeList.push("P" + period + "_" + timeSpentMs);
        regionList.push("P" + period + "_" + config.name);

        // Update main sequence - add both region name and time
        mainSequence.push("P" + period + "_" + config.name + ":" + timeSpentMs);

        // Write to the THREE embedded data fields
        Qualtrics.SurveyEngine.setEmbeddedData("timeList", timeList.join(","));
        Qualtrics.SurveyEngine.setEmbeddedData("regionList", regionList.join(","));
        Qualtrics.SurveyEngine.setEmbeddedData("mainSequence", mainSequence.join(","));

        // Clean up
        delete regionStartTimes[regionId];
    }
}

// Dynamically generate show/hide functions for each region
Object.keys(regionConfig).forEach(function(regionId) {
    var config = regionConfig[regionId];

    window["show" + config.name] = function() {
        showRegion(regionId, config.hiddenContent);
    };

    window["hide" + config.name] = function() {
        hideRegion(regionId);
    };
});
```

Notes: Paste this entire code chunk before the start of the custom timing functions—i.e., before *addOnload*—in the question containing the mouse-tracking.

Step 5 – Modify the JavaScript with Custom Content

You will need to modify the highlighted JavaScript (from Step 4) to reflect your specific study design. All modifications occur within the regionConfig configuration object, which is a data structure containing one or more key-value mappings. The structure is defined as { Key 1: {name: " ", hiddenContent: " ", revealedContent: " "}, … , Key *k*: { name: " ", hiddenContent: " ", revealedContent: " "} } for a design with *k* mouse-tracked regions. The purpose of this configuration object is to create *k* unique show() and hide() functions, each with their own behaviors.

Within the regionConfig object, the *k* keys must be named exactly the same as the *ids* assigned to each region in your HTML code. For example, the JavaScript keys (within regionConfig) in Figure W8 are named exactly the same as the two HTML *ids* used within the <td> tags Figure W6.

Figure W8: Updating Mouse-Tracked Content.

```
var regionConfig = {
    "Region1": {
        name: "R1",
        hiddenContent: "Region 1 Test",
        revealedContent: ""
    },
    "Region2": {
        name: "R2",
        hiddenContent: "Region 2 Test",
        revealedContent: ""
    }
};
```

Notes: The regionConfig object must be updated to reflect the correct number of keys (corresponding to the number of mouse-tracked regions) and their specific values for *name*, *hiddenContent*, and *revealedContent*.

In addition to updating the number of key-value mappings and ensuring each key matches an HTML *id*, the values for *name*, *hiddenContent*, and *revealedContent* must also be defined.

The *name* value should reflect a clear and systematic naming convention corresponding to the different mouse-tracked regions. By default, we use a numbering convention, such that keys are named R1, …, R*k*. These names must exactly match the name portion of the unique show() and hide() functions in HTML. For example, in Figure W6, function names showR1(), hideR1(), showR2(), and hideR2() correspond to the name values of "R1" and "R2" in regionConfig (Figure W8). Should you choose to adopt a different naming convention, you must ensure your HTML function calls (e.g., showMyRegion_1()) matches the name value ("MyRegion_1"). Whatever naming convention you use will also be used to index the mouse-tracked regions in your Qualtrics embedded data, specifically, within the *regionList* and *mainSequence* variables.

The *hiddenContent* value is the concealed information that is only available during a mouse hover. The value of *hiddenContent* must take the form of a text string that can be passed to HTML. The example in Figure W8 uses plain text; however, anything that can be rendered in HTML (e.g., simple text, but also images, or styled text) can technically be contained here. Additional details for modifying the HTML include images or styled text are provided in Step 10.

The *revealedContent* value is the information that is displayed by default (i.e., without a mouse hover). You may set this value as empty by providing an empty string (i.e., " "). You can also provide any text string, as with *hiddenContent*.

Step 6 – Updating the JavaScript addOnload Function

You will need to modify the Qualtrics timing functions with additional code snippets. These timing functions ensure that certain portions of code do not execute too early, or out of order. We recommend modifying the *addOnload* function to include the call to initialize the various regions. This is implemented by inserting the single line in Code W4. A completed Step 6 is visualized in Figure W9.

Code W4: JavaScript Snippet to Initialize Content.

```
initializeRegionContent();
```

Notes: Paste code snippet in the *addOnload* function of the question with mouse-tracking.

Figure W9: The addOnload Timing Function After Adding Code Snippet

```
Qualtrics.SurveyEngine.addOnload(function() {

    // Initialize the region content from JavaScript
    initializeRegionContent();

});
```

Notes: Adding this line to the *addOnload* function will ensure the call to initialize the page (with the default revealed content) occurs after region structure exists.

Step 7 – Updating the JavaScript addOnReady Function to Update the Period Count

There are two code snippets that should be placed in the *addOnReady* function. The first section of code increments the *period* variable. While this bit of code could be omitted for a single-shot design, we recommend including it in all studies. In a repeated-measure design (e.g.,

either multiple mouse-tracked questions separated by page breaks, or the same question iterated by a loop and merge), updating this *period* variable is essential. You should paste the following code—without modification—into the start of the *addOnReady* function (Code W5).

Code W5: JavaScript for addOnReady Function.

```
var nextPeriod = period + 1;
Qualtrics.SurveyEngine.setEmbeddedData("period", nextPeriod);
```

Notes: Paste this in *addOnReady* function of question with mouse-tracking.

Step 8 – Updating the JavaScript addOnReady Function with Error-Catching Code

As a best-practice, we also recommend including an additional code chunk to double-check the mouse-tracked table (or tables) are behaving as expected (Code W6). Specifically, this code operates at the level of HTML tables (rather than table cells, like the *onmouseenter* and *onmouseleave* defined in Step 3). The purpose of this code is to listen for any instance in which a mouse leaves a table and then ensure every cell is in the hidden state. The reason this code is important is because very fast or erratic mouse movements into and out of a region may potentially cause the show() and hide() functions to complete out of order. While this potential problem is minimized with the slight timing delay from Step 1 (the *delayInMs* variable), our error-catching code provides another layer of protection against potential malfunctions. Add the following code chunk into the *addOnReady* function after the previous code (Code W6).

Code W6: JavaScript for Error-Catching, Placed in addOnReady Function.

```
    var tables = document.querySelectorAll('table');
    tables.forEach(function(table) {
        table.addEventListener('mouseleave', function() {
            // Anytime a mouse leaves a table, ensure all regions are hidden
            Object.keys(regionStartTimes).forEach(function(regionId) {
                if (regionStartTimes[regionId]) {
                    window["hide" + regionConfig[regionId].name]();
                }
            });
        });
    });
});
```

Notes: Paste this in *addOnReady* function of question with mouse-tracking.

After completing Steps 6-8, you should have code in the *addOnload* and *addOnReady* functions (and nothing in the *addOnUnload* function), as shown in Figure W10.

Figure W10: Full View of JavaScript Timing Functions After Code Additions.

```
// Initialize content when page loads
Qualtrics.SurveyEngine.addOnload(function() {

    // Initialize the region content from JavaScript
    initializeRegionContent();

});

Qualtrics.SurveyEngine.addOnReady(function() {

    // Set the new value
    var nextPeriod = period + 1;
    Qualtrics.SurveyEngine.setEmbeddedData("period", nextPeriod);

    // Adding an additional failsafe listener that makes sure information in the table is concealed (whenever a mouse leaves ANY region)
    var tables = document.querySelectorAll('table');
    tables.forEach(function(table) {
        table.addEventListener('mouseleave', function() {
            // Anytime a mouse leaves a table, ensure all regions are hidden
            Object.keys(regionStartTimes).forEach(function(regionId) {
                if (regionStartTimes[regionId]) {
                    window["hide" + regionConfig[regionId].name]();
                }
            });
        });
    });
});

Qualtrics.SurveyEngine.addOnUnload(function() {

});
```

Step 9 – Creating the Custom Next Button

You will likely need to create your own next button (or buttons) to ensure any page-advance mechanics do not mechanically or systematically favor one mouse-tracked region over another. We expect the custom next button will be placed on the page before the mouse-tracked page (thus ensuring the participant's mouse is positioned at a target location, prior to exposure to the mouse-tracked ROIs). Creating a custom next button involves three steps, spanning HTML, JavaScript, and CSS.

First, create a new question in Qualtrics of the "Text / Graphic" type (i.e., a question without any responses or form fields). Using the *HTML View*, paste the following code snippet to create an HTML class and an HTML button (Code W7).

Code W7: HTML for Custom Next Button.

```
<div class="button-container">
   <button type="button" class="custom-next-button" id="customNextBtn">
      &rarr;
   </button>
</div>
```

Notes: Paste using *HTML View* in the question where you intend to have a custom next button.

The button displays an arrow ("→"), which is referenced by the Unicode "→" and resembles Qualtrics' next button. You may choose to update the button contents ("→") by replacing "→" with any other text.

Second, you will need to insert custom JavaScript into the question containing the HTML button (Code W8). The purpose of the code is to (i) suppress Qualtrics's default next button and (ii) assign functionality to your HTML button. Paste the following code into the *addOnReady* function, noting that the variable name within the jQuery call ("#customNextBtn") must exactly match the HTML button id (id="customNextBtn").

Code W8: JavaScript for Custom Next Button.

```
// This portion hides the default next button (only on this page)
this.hideNextButton();

  var that = this;
    // Get reference to the custom button (HTML button id must match this!)
    var customButton = jQuery("#customNextBtn");
    customButton.on('click', function() {
        that.clickNextButton();
    });
```

Notes: Paste in the *addOnReady* function of the question where you intend to have a custom next button.

Third, you will want to modify the CSS to update the styling of your custom next button. The most straightforward approach is to define the HTML classes that wrap your HTML button. To start, we recommend pasting the following CSS code into the "Custom CSS" field (Code W9), contained with the "Look and Feel" section (*Look and Feel* → *Style* → *Custom CSS*).

Code W9: CSS for Button Styling.

```
/* Button styling */
.custom-next-button {
/* Modify this background-color with your theme color ("primary color") in look and feel */
    background-color: #007AC0;
    color: #FFFFFF !important;
    padding: 12px 24px;
    font-size: 16px;
    font-weight: bold;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    transition: background-color 0.3s ease;
    margin-top: 20px;
}
/* Hovering aesthetics */
.custom-next-button:hover {
    filter: brightness(0.90);
}
.custom-next-button:active {
    filter: brightness(0.85);
}
/* Button position */
.button-container {
    text-align: center;
    margin-top: 30px;
}
```

Notes: Paste this code into *Look and Feel* → *Style* → *Custom CSS*.

Using this code, you can modify (i) how the button looks (.custom-next-button{…}); (ii) the interaction aesthetics during button hovers and clicks (.custom-next-button:hover {…} and .custom-next-button:active {…}); and (iii) the button position on the page (.button-container {…}). For example, you might want to modify your next button's color to match your specific Qualtrics theme. To do this, you could update the *background-color* variable contained within the styling for *.custom-next-button{…}* to match whatever "Primary Color" is set for your theme in Look and Feel (*Look and Feel* → *Style* → *Primary Color*).

After creating the new Qualtrics question with custom HTML, JavaScript, and CSS, you will have a question that can be moved or copied anywhere you need a custom next button. You can treat this as a standalone question (i.e., the only content within a set of page breaks) or you can add this question alongside another question (e.g., one with mouse-tracking) within a set of page breaks (Figure W2).

Step 10 (Optional) – Modifying the Type of the Hidden or Revealed Content

Passing HTML through JavaScript

There may be instances in which you prefer to modify the content of mouse-tracked regions to include elements beyond plain text. As an example, the simple string of "Region 1 Test" (Figure W8) could be modified to present the same text as bolded ('Region 1 Test'). Alternatively, you may want to include a graphic during a dwell within a region of interest ('<img src="https://www.ama.org/wp-content/themes/ama/assets/images/ama-logo-trans.png"> Region 1 Test'). For any such cases, the best-practice is to write your HTML within JavaScript (contained with the regionConfig object, as discussed in Step 5).

Placing HTML in JavaScript requires two additional steps. First, you should wrap any HTML in the following styling code to prevent conflicts with the existing *onmouseenter* and *onmouseleave* code: (your content is placed between the <span… > and tags). Second, ensure you are using single quotations (') to wrap the JavaScript string (i.e., use ' marks as the outermost characters). This will ensure any special characters (e.g., double quotation marks) contained within the HTML character string do not require special handling (i.e., escaping with a backslash). For example, the following JavaScript string (of HTML) would be read properly ('Region 1 Test'). However, the code would not execute properly if the outermost single quotes were changed to double quotes (").

Image caching

For any versions of QMT containing images, we recommend adding additional JavaScript code to ensure the images are pre-loaded into the participant's browser's cache. Image caching (partial pre-loading) ensures the images load quickly. This is especially important for mouse-tracking contexts, because slow-loading images could inadvertently require longer dwells (for mechanical, not attention-based reasons). The easiest way to implement image caching is to add this portion of JavaScript to either (i) another question that occurs on a prior page to the mouse-tracked content or (ii) in the area before the timing functions (i.e., the location labeled "Most of the mouse-tracking JavaScript goes here" in Figure W7). The only requirement is to ensure this code executes prior to calling the code that initializes all regions (and pulls data from regionConfig) (Code W10).

Code W10: JavaScript for Image Caching.

```
// Preload all specified images
var imagesToPreload = ["https://www.ama.org/wp-content/themes/ama/assets/images/ama-logo-trans.png", "ANY OTHER URLS GO IN THIS ARRAY"];
imagesToPreload.forEach(function(imageUrl) {
    var img = new Image();
    img.src = imageUrl;
    // The browser will cache these automatically!
});
```

Notes: Paste this code into the JavaScript *addOnReady* function of a question that occurs prior to the page containing mouse-tracking.

Step 11 (Optional) – Data Transformation After Data Collection

After participants complete the survey, all mouse-tracking data will be contained within the three variables *timeList*, *regionList*, and *mainSequence* that were defined in Step 1. These variables contain character strings, requiring additional steps to access the mouse-tracking data. We recommend conducting all data transformation *outside* of Qualtrics, using your preferred statistical software. Here, we provide the R code to transform the embedded data character strings into a long dataset (Code W11). The resulting data structure contains a row for each mouse-tracked observation, with participant, period, region, and time information:

Code W11: R Code to Clean and Transform Mouse-Tracking Data.

```
library(tidyverse)

## We refer to the wide (raw) data as df
df <- read.csv("qmt_data.csv")

## This could also be set to a lab ID, workerId, PROLIFIC_PID, etc.
df$subject_id <- c(rep(1:nrow(df)))
## Unravel the char string into a long data format
df_long <- df %>%
  ## Separate the triple-asterisk-delimited observations into individual rows
  separate_rows(mainSequence, sep = "***") %>%
  ## Trim any whitespace
  mutate(mainSequence = trimws(mainSequence)) %>%
  ## Extract period, region, and time data
  extract(mainSequence,
          into = c("period", "region", "time"),
          regex = "(P\\d+)_(\\w+):(\\d+)") %>%
  ## Just making sure this is numeric (and not character)
  mutate(time = as.numeric(time)) %>%
  ## Select final columns
  select(subject_id, period, region, time)
```

Notes: This R code requires download and install of the "tidyverse" package and assumes the raw data (with only a single header row) are in a .csv file titled "qmt_data.csv"

Step 12 (Optional) – Installing the Debugging Tool

Throughout the development of your QMT survey, you may occasionally find it useful to use our debugging tool. You can access the tool by including a few additional lines of JavaScript and then turning on the debugging mode through embedded data.

First, you will need to insert the following code into the *addOnReady* function (Code W12). We recommend placing this at the start of *addOnReady* (e.g., right before declaring the var *nextPeriod*). This bit of code calls an externally hosted JavaScript debugging tool and loads it into your browser.

Code W12: JavaScript to Install Debugging Tool.

```
var debugMode = parseInt("${e://Field/debugMode}") || 0;

if (debugMode == 1) {
  var script = document.createElement('script');
  script.src = 'https://cdn.jsdelivr.net/gh/QMT-code/QMT/QMT_debugging.js';
  document.head.appendChild(script);
}
```

Notes: Paste at the start of the *addOnReady* function of the question containing mouse-tracking.

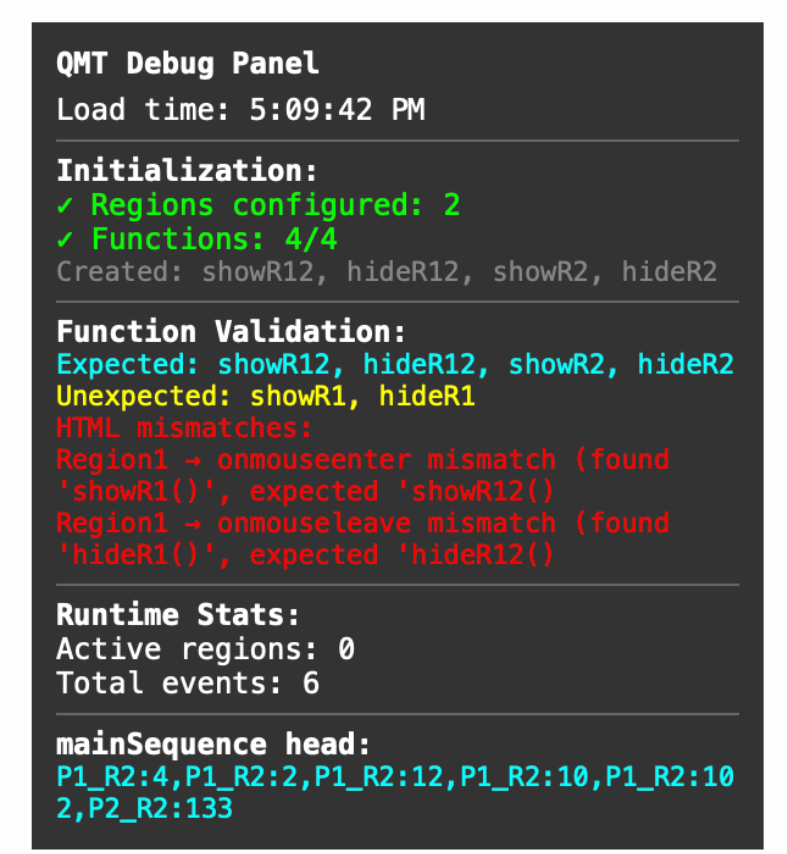
Second, you will need to add one more variable to *Embedded Data* (accessed through the *Survey Flow*). This variable is called *debugMode*. If *debugMode* is undefined (i.e., not declared in *Embedded Data*) or takes on any value other than "1," then the debugger will be turned off. If you assign a value of "1" to *debugMode* in embedded data, then the debugging tool will appear when the survey is running (e.g., previewed *or* run through an anonymous link).

As shown in Figure W11, the debugger loads a panel that summarizes the behavior of the JavaScript code. After verifying JavaScript is correctly pulling the time (necessary for the time-keeping functionality), the debugger confirms the number of configured regions (from regionConfig), as well as the number of observed and expected functions created for the show() and hide() functionality. The debugger provides the names of the created show() and hide() functions, and validates whether these function names match the HTML. In Figure W11, the debugger identifies that two of the function names (showR12() and hideR12()) are different than expected (showR1() and showR2()). In this example, there is a discrepancy—a likely typo—between the naming convention used in HTML and JavaScript's regionConfig object. In the lower panels of the debugging tool, the Runtime Stats monitor whether JavaScript recognizes

mouse-hover events. The mainSequence head panel provides the real-time contents of the mouse-tracking data in their character string format.

To turn debug mode off, change the value of *debugMode* in embedded data to anything other than "1" (e.g., set it to "0").

Figure W11: The Custom Debugging Tool.



Notes: The custom debugging tool provides real-time updates about the question's JavaScript code, including the number of regions initialized, as well as the number—and names—of custom show() and hide() functions created. The tool validates the JavaScript function names against the expected names from HTML (highlighting any discrepancies in red text, as shown). The debugging panel also provides real-time updates regarding mouse hovers, as well as the contents of embedded data.

Troubleshooting

My tables are showing up but are not responding to mouse movements.

1. Check the *HTML View* of the question to ensure the mouse-tracked regions contain the *onmouseenter* and *onmouseleave* event listeners. (These can be unknowingly erased by Qualtrics if you modify the mouse-tracked question without using *HTML View*.)
2. Check the JavaScript to ensure your code is placed before the *addOnload* timing function. All code pertaining to the show() and hide() functionality—as well as the regionConfig object—must be defined beyond the timing functions to have a global scope.
3. Confirm the naming conventions used in HTML (the "id" and the names of the functions called by *onmouseenter* and *onmouseleave*) match the keys and values in regionConfig. Specifically, if there is an id = "Region1" (e.g., <td id = "Region1" …) then the corresponding JavaScript key must be "Region1" (e.g., "Region1" : {name: …} …) (Figure W8). Furthermore, the unique function names assigned to *onmouseenter* (e.g., "showR1()") and *onmouseleave* (e.g., " hideR1()") must correspond to the name assigned for that key. In the examples provided in Figure W6 and Figure W8, this corresponds to the JavaScript name "R1" (within the regionConfig object and the "Region1" key).
4. Confirm you have assigned valid JavaScript strings to the values of *hiddenContent* and *revealedContent*.
5. Confirm you have included the call to initialize the regions in the *addOnload* function (Code W4).
6. Launch the debugger by completing Step 12 (Figure W11).

My mouse-tracking code keeps disappearing.

1. Ensure there is at least some plain text contained within your Qualtrics question. The reason is because when Qualtrics refreshes a question, which occurs when the question contents are clicked on—even if they are not modified—it attempts to "clean up" questions that appear empty. You can protect your question from being "cleaned up" by including some regular text, as shown in Figure W6.
2. Ensure that you are only using *HTML View* to modify the contents of a mouse-tracked question. Even if you are attempting to modify plain text, you must do so through the HTML view. Note: you do not need to wrap plain text in HTML tags or do anything special; if you are in the *HTML View*, you may still write plain text.