Behav Res (2015) 47:620–631
DOI 10.3758/s13428-014-0493-8

Visual DMDX: A web-based authoring tool for DMDX, a Windows display program with millisecond accuracy

Pablo Garaizar · Ulf-Dietrich Reips

Published online: 10 June 2014

Abstract DMDX is a software package for the experimental control and timing of stimulus display for Microsoft Windows systems. DMDX is reliable, flexible, millisecond accurate, and can be downloaded free of charge; therefore it has become very popular among experimental researchers. However, setting up a DMDX-based experiment is burdensome because of its command-based interface. Further, DMDX relies on RTF files in which parts of the stimuli, design, and procedure of an experiment are defined in a complicated (DMASTR-compatible) syntax. Other experiment software, such as E-Prime, Psychopy, and WEXTOR, became successful as a result of integrated visual authoring tools. Such an intuitive interface was lacking for DMDX. We therefore created and present here Visual DMDX (http://visualdmdx.com/), a HTML5-based web interface to set up experiments and export them to DMDX item files format in RTF. Visual DMDX offers most of the features available from the rich DMDX/DMASTR syntax, and it is a useful tool to support researchers who are new to DMDX. Both old and modern versions of DMDX syntax are supported. Further, with Visual DMDX, we go beyond DMDX by having added export to JSON (a versatile web format), easy backup, and a preview option for experiments. In two examples, one experiment each on lexical decision making and affective priming, we explain in a step-by-step fashion how to create experiments using Visual DMDX. We release Visual DMDX under an open-source license to foster collaboration in its continuous improvement.

Keywords DMDX · HTML5 · Internet-based research · JSON · Web technologies

Several software packages are available to conduct psychological experiments under strict timing requirements. These packages can be categorized by different dimensions: (1) general versus specific purpose, (2) single-platform versus multiplatform, (3) online versus offline, and (4) commercial versus noncommercial (see Mathôt, Schreij, & Theeuwes, 2012). Because of this wide variety, researchers are often forced to learn a large number of configuration options and specific programming languages. Frequently, one program may have fewer options than another, more powerful program, but offer a small, indispensable set of extra features that the latter program does not implement. In addition, the chances of technical errors tend to increase with the size and power of software packages, and such errors may go undetected by researchers who do not have an extensive knowledge of computer science.

Given the wide variety of alternatives, many researchers choose to use a tool that has the endorsement of having been used in many previous studies. Such is the case with DMDX. The main reasons for the success of DMDX in experimental psychology are its price (it can be downloaded and used at no cost) and the existence of a community of researchers who are using DMASTR, a set of experiment utilities developed at Monash University (Australia) beginning in 1975. This software was improved by various contributors over the years and ported to Windows 95/98/Me by Jonathan Forster in 1997. Despite the initial reluctance

P. Garaizar
Faculty of Engineering, University of Deusto, Bilbao, Spain

U.-D. Reips
Department of Psychology, University of Konstanz, Konstanz, Germany

U.-D. Reips
Ikerbasque, Basque Foundation for Science, Bilbao, Spain

P. Garaizar (✉)
Deusto Institute of Technology (DeustoTech), Universidad de Deusto, Avda. Universidades 24, 48007, Bilbao, Spain
e-mail: garaizar@deusto.es

regarding the suitability of Windows NT and Windows 2000 to directly access video, audio, and timing hardware (see, e.g., Krantz, 2000), Forster and Forster (2003) devised a way to port the old DMASTR to Windows 2000/XP (or later versions) through the use of DirectX and multimedia timers. This is the reason for the name DMDX—it is a combination of DMASTR and DirectX.

Benefits and pitfalls of DMDX

The large size of the DMDX user community is not motivated only by its long history. Among other important factors, four stand out: price, support, accuracy, and power. First, DMDX can be downloaded and used free of charge. All updates are being made available from the DMDX home page to the community, as they are developed. Moreover, DMDX's main developers offer free technical support via e-mail. Third, DMDX uses the most appropriate multimedia technologies and timing mechanisms to maximize accuracy and precision during the stimuli presentation and gathering of user responses. Auxiliary tools for fine tuning (such as TimeDX) are provided with DMDX for achieving this goal, even on commodity hardware (i.e., regular PCs). Fourth, DMDX is powerful. Researchers have almost unlimited options to create experimental designs and procedures by using more than 400 keywords to create loops, conditional branches, or scrambled lists of trials.

DMDX is powerful and accurate. Forster and Forster (2003) measured the validity of DMDX presenting stimuli, reporting zero errors after 100 tests. With regard to response times, they found a standard deviation of 0.53 ms (i.e., submillisecond precision). DMDX achieves these results by two different methods. First, DMDX uses the real-time clock system—which is independent of the system load—to minimize errors in the presentation of stimuli. Since waiting times are known, the thread can be idle until a moment shortly before the screen refresh. After the timer period expires, DMDX actively checks whether the screen refresh has occurred. If DMDX fails to detect the screen refresh signal through a timer (properly calibrated through TimeDX), it will infer that the screen refresh already happened and reset the timer according to this assumption. Second, to avoid possible errors recording user interaction, DMDX polls input devices every millisecond, using a multimedia timer.

However, there are also some drawbacks to using DMDX. Although DMDX is an application with a graphical user interface (GUI), the process to design, develop, and carry out an experiment differs little from working with commands directly. At a user level, the biggest advantage of DMDX (i.e., its compatibility with DMASTR and its associated power) is also its main problem, because the syntax of the configuration files is far from intuitive for those using the tool for the first time. Mastering DMDX syntax is not easy, mainly because of the sheer number of terms available. DMDX currently offers 414 keywords, including 224 synonyms, to define subroutines and other aspects of an experiment. Moreover, it is noticeable that DMDX uses a 30-year-old syntax: More than 75 different branching keywords combined with global counters are needed to control the flow of an experiment, instead of parsimoniously combining a small set of reserved terms such as *if, for,* or *while* with Boolean expressions.

There are two different parts in a DMDX "item file": *general parameters* and *items*. The general parameters are placed at the beginning of the item file and define global settings that apply to all parts of an experiment, unless explicitly noted otherwise. For example, screen resolution of the experiment, time between items, input devices, and so on are defined here. After the general parameters are set, it is necessary to configure the items (i.e., the trials of the experiment).

Table 1 shows a simple example of a DMDX item file with two items (one per line) that we will explain here to demonstrate the complexity of creating experiments with DMDX. The first character of each line indicates whether the expected response is positive (+) or negative (–). It is also possible to configure items that require no response or items for which any answer is correct. Then a numerical value acts as item identifier. If two items have the same identifier, only the response to the latter is stored. When zero is used as the item identifier, DMDX assumes it is not an experimental item, but an instructions message. Then, as many frame elements as are necessary are defined, using "/" as the frame separator. Every frame may contain text (delimited by double quotes) or references to images, videos, or audio files. The duration of each frame can be configured using the "%" symbol followed by the number of ticks that the frame should be presented. (If no specific duration is defined, the standard duration defined as a general parameter is used.) The "*" symbol indicates the point at which the timer measuring user responses is started. The ";" symbol marks the end of the item definition. Thus, in the example shown in Table 1, two items are defined to affect user performance in classifying a word as positive or negative after the brief presentation of another word. The first item (001) requires a positive response (+). Four frames are defined for this item: (1) a frame in which the "+" symbol is

Table 1 Items definition in a DMDX / DMASTR item configuration file

```
+001 "+" %60 / "HOPE" %15 / * "LOVE" %120 / ;
-002 "+" %60 / "LOVE" %15 / * "SAD" %120 / ;
```

displayed in the center of the screen for 60 ticks (1,000 ms at 60 Hz), (2) a frame in which the word "HOPE" is shown for 15 ticks (250 ms at 60 Hz), (3) a frame in which the word "PEACE" is displayed for 120 ticks (2,000 ms at 60 Hz), and (4) an empty frame. DMDX starts measuring a participant's response time at the beginning of the third frame (because of the "*" symbol). The second item (002) is very similar, except for the expected response (positive, in this case). As can be seen, the definition of each item requires knowledge of a very specific syntax. The complexity of this syntax increases when advanced functionality is required (e.g., conditional branches, loops, automatic generation of items, etc.).

Editing a DMDX item file is not an easy task. Moreover, most DMDX users are not really interested in coding experiments, but in running them. This results in many instances in which DMDX users just copy, paste, and modify experiments previously created by other researchers, resulting in functional but very difficult to understand code. For instance, DMDX offers very advanced features for the iteration and shuffling of items. Using counters in DMDX, it is possible to repeat a set of items as many times as necessary. There are also plenty of alternatives to scramble items on DMDX keywords, such as Scramble, ScrambleSeed, and Grouping. Backslashes can also be used to split the item file into two or more parts, with different scrambling setups. However, both features require a thorough knowledge of the syntax of DMDX and are often misused by researchers. At the same time, several advanced DMDX users (Woods, Forster, Curtin, and others offer tutorials aimed at beginners, which is to say that they cover only the basics. With the help provided in these tutorials, novice DMDX users are able to create their own experiments from scratch.

DMDX is therefore a highly recommended experimental tool, both in an economical sense (because it is free) and a technical one (because it is very precise and accurate). Further, DMDX is well-supported and has an active user community. However, its learning curve is still very steep for beginners. For this reason, we developed Visual DMDX, a web-based authoring tool that makes it possible to create DMDX item files within minutes. In the following section, we discuss Visual DMDX's main features.

Visual DMDX: A web-based authoring tool for DMDX

Experiment software such as E-Prime (Schneider, Eschman, & Zuccolotto, 2002), Psychopy (Peirce, 2007), and WEXTOR (Reips & Neuhaus, 2002) became successful as a result of integrated authoring tools that help the experimenter design and conduct experiments in intuitive ways. Ideally, there is no need to deal directly with the syntax in designing and conducting experiments. DMDX would thus benefit from an intuitive interface.

> Modern software is often programmed as a web service
> "because the functionality of Web browsers is less dependent on the operating system (sometimes they are even referred to as being platform independent), all who access a Web service are likely to see and experience almost the same interface […]. Web services spare the user from upgrading and updating, since this is done by the Web service administrators at the server. Nothing is installed on the user's computer, saving space and time." (Reips & Lengler, 2005, p. 287).

Visual DMDX is programmed as a web service to complement the suite of web-based tools available for research, many of which were published in *Behavior Research Methods* (FactorWiz and SurveyWiz: Birnbaum, 2000; ReCal and ReCal OIR: Freelon, 2010, 2013; Social Lab: Garaizar & Reips, 2013; Form Processor: Göritz & Birnbaum, 2005; VAS Generator: Reips & Funke, 2008; iScience Maps: Reips & Garaizar, 2011; Web Experiment List: Reips & Lengler, 2005; Web Experimental Psychology Lab: Reips, 2001; WEXTOR: Reips & Neuhaus, 2002; Dynamic Interviewing Program and User Action Tracer: Stieger & Reips, 2008, 2010).

Structure and design

Our main purpose with Visual DMDX is to provide a *web service* that facilitates the process of creating DMDX item files. Following good design for web-based research tools (Reips, 2010; Reips & McClelland, 2013) we decided to develop a web application based on open standards such as HTML5, CSS, and JavaScript, so it could be used from any platform (e.g., Windows, Mac OS, GNU/Linux, iOS, Android, etc.) without installing add-ons or plug-ins. Since one of the initial requirements of the tool was to be able to work from anywhere at any time, Visual DMDX does not require user accounts and passwords. Instead, it uses *unique URLs* to edit each experiment. These URLs are defined when creating the experiment and can easily be shared with all members of the research team. The format of Visual DMDX URLs is thus similar to those used on sites such as *YouTube*, where each video is identified by a string of alphanumeric characters. Six alphanumeric characters are used to identify each experiment in Visual DMDX (see Fig. 1A), for example, http://visualdmdx.com/edit/7qvofp (where the identifier *7qvofp* points to the experiment). Furthermore, because Visual DMDX allows researchers to store successive versions of the same experiment, each version will have its own URL derived from the URL of the experiment. Thus, the URL http://visualdmdx.com/edit/7qvofp/1 is the URL of the

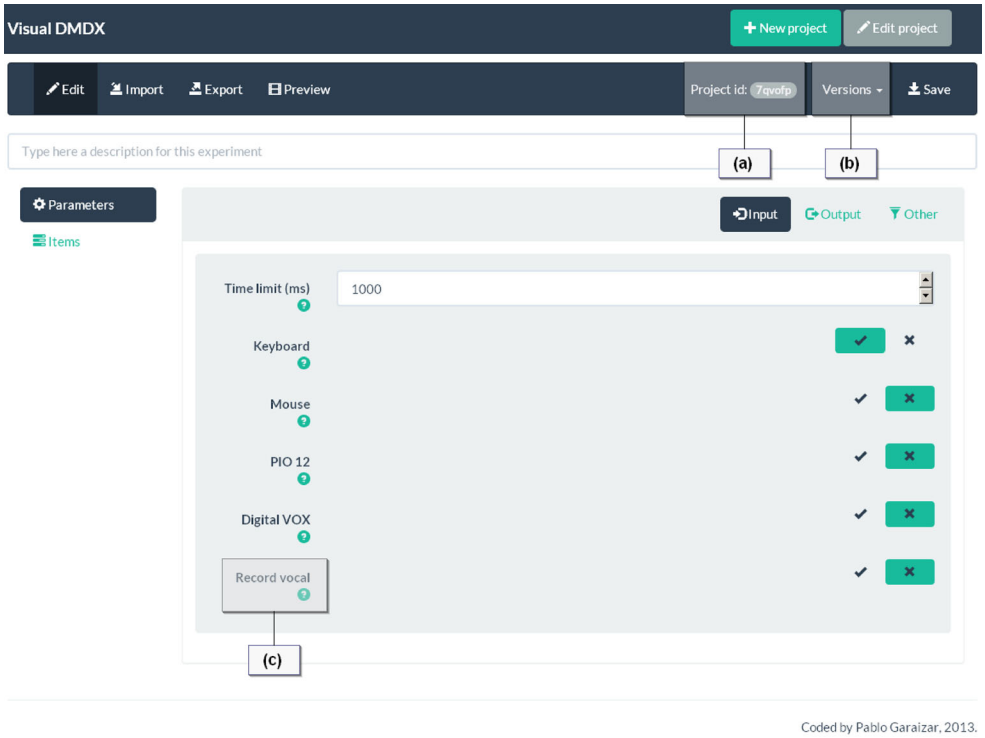

Fig. 1 Visual DMDX user interface: (A) unique identifier of the experiment; (B) dropdown list of the versions of the experiment; (C) link to the contextual help of the keyword related to this parameter

first version of the experiment 7qvofp; http://visualdmdx.com/edit/7qvofp/2 corresponds to the second version of the same experiment; http://visualdmdx.com/edit/7qvofp/3 corresponds to the third, and so on. This naming convention provides access to the history of each experiment. From the user interface of Visual DMDX it is also possible to access each of the previous versions from the "Versions" dropdown menu in the header (see Fig. 1B).

Since Visual DMDX is designed to be used mostly by novice users, it does not require any knowledge of the syntax used in DMDX item files. On the contrary, Visual DMDX users only have to *define general parameters via simple web controls* (i.e., text fields, checkboxes, dropdown lists, buttons, drag-and-drop sortable lists, tabs, modal dialog boxes, etc.) and create new items using a *drag-and-drop editor*. Following the recommendations of the previously mentioned manuals by Forster and others, Visual DMDX currently supports 43 keywords (18 synonyms) in DMDX. For the sake of simplicity, Visual DMDX assumes that DMDX runs under an English version of Microsoft Windows. (Users should be aware of possible configuration issues related to DMDX and other language versions of Windows; e.g., the "keyboard" input device is called "clavier" in French and "teclado" in Spanish and thus DMDX will not run under respective versions of Microsoft Windows if the device is not renamed manually in the DMDX item file.) These design decisions have two advantages: (1) keeping a simple user interface, (2) covering the most common DMDX features in simple experiments. In addition, general parameters are provided with contextual help that shows the underlying keywords involved and their different options (see Fig. 1C).

Item editor

The most remarkable feature of Visual DMDX is the item editor, with which users can create new instructions items, stimuli presentation items, and loops. The *instructions items* are the simplest types of items. They are used to define an informative text that will be displayed to the participant until the spacebar is pressed. All items allow users to define a description that will not be exported to the DMDX syntax, but may serve to identify different parts of the experiment (see Fig. 2).

Stimuli presentation items are responsible for controlling the trials of the experiment. In these items, researchers

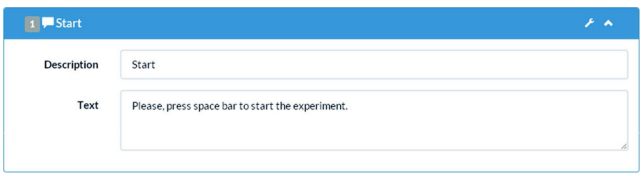

Fig. 2 Instructions item

define details such as: (1) what stimuli to present to participants, (2) how long to present stimuli, (3) when to start measuring response times, and (4) what response to expect (i.e., positive, negative, none, or any). Within each stimuli presentation item, researchers can define several types of stimuli: blank, text, image (BMP or JPG), audio, and video (see Fig. 3). Adding a stimulus necessitates definition of its presentation duration and other parameters within the item, depending on the type of stimulus—for example, path to the file in the case of multimedia stimuli, horizontal and vertical position on the screen for visual stimuli, left or right channels for audio stimuli, and so forth. It is possible to change the order of the stimuli simply by clicking on one stimulus and dragging it to the desired position on the stimulus list of the item. The same applies to items: to change the order in which they will be exported to DMDX, users simply have to drag an item to the desired position. Considering that the definition of a trial (i.e., a stimuli presentation item) is a laborious task, Visual DMDX provides the ability to clone an existing item.

Loop items are exclusive to Visual DMDX and go beyond options available in DMDX. As mentioned in the previous section, DMDX provides advanced features for the iteration and shuffling of items. However, using these features is beyond the reach of DMDX novice users because they have to understand how to set and update global counters and use the proper branching keyword (from a set of more than 75 branching keywords). Therefore, we propose an alternative method: loop items. When a researcher creates a loop item, she must place it immediately before the first item to be repeated. Then, she must indicate which item is the last of the loop, how many iterations should be performed, and whether the items in the loop must be scrambled at each iteration. In this way it is possible to make simple loops, shuffling (i.e., one iteration loop with scrambling), and loops with scrambling (Fig. 4).

Because loop items do not really exist in the DMDX syntax, loops are "unrolled" when Visual DMDX exports an experiment to DMDX. This means that Visual DMDX will create as many new items as needed (e.g., if we have a

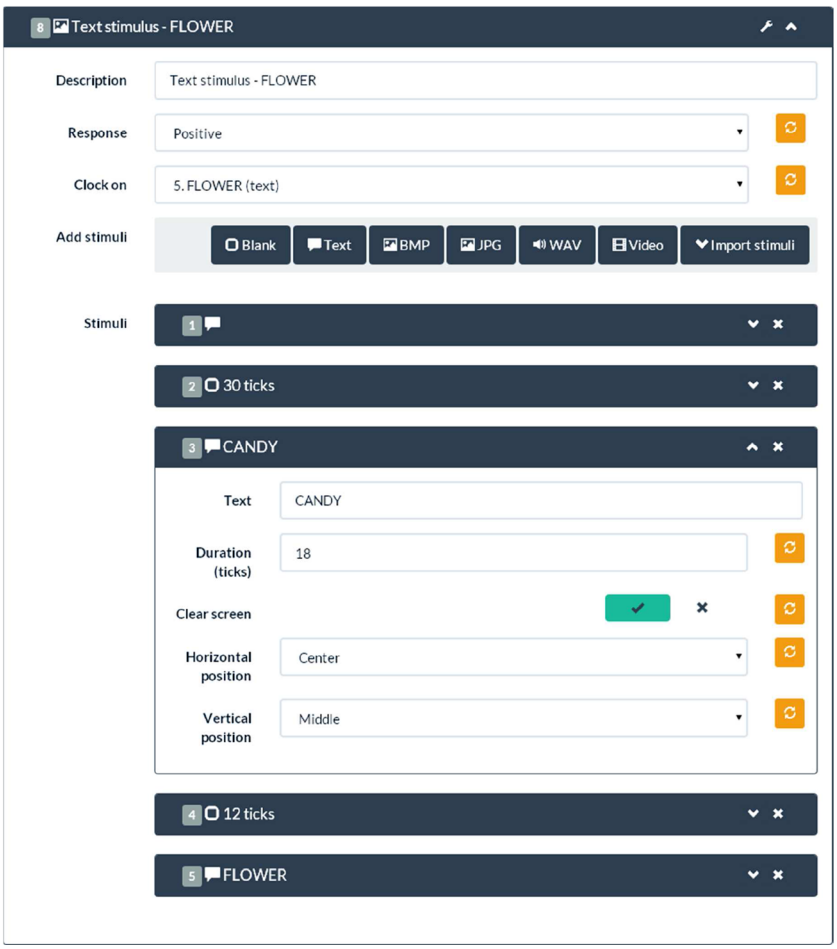

Fig. 3 Stimuli presentation item

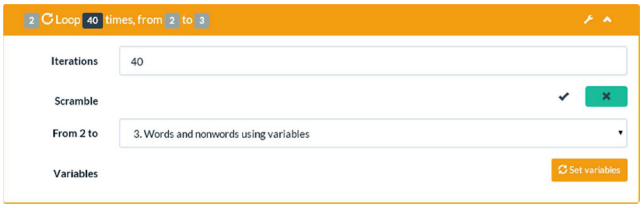

Fig. 4 Loop item

five-iteration loop for three items, five copies of those three items will be created, making a total of 15 items). In the case of loops with scrambling, each time the researcher asks Visual DMDX to export her experiment to DMDX, a new order will be generated. Nevertheless, it is always possible to combine the output of Visual DMDX with the DMDX scrambling keywords discussed above (i.e., Scramble, Grouping, and ScrambleSeed). Moreover, creating complex nested structures is straightforward using loop items; for example, a three-iteration loop for six items and an inner loop within these six items that repeats the last two items twice would generate a list of 24 items: 3 × (4 + 2 × 2).

Loop items also enable researchers to create a list of variables that may take different values in each iteration of the loop. To facilitate the definition of these lists of variables, Visual DMDX provides a variable editor. In the cases in which the list of variables and/or values is long, researchers can use the "Import variables from CSV" functionality of the editor. To use one of these variables, researchers must indicate the name of the variable in brackets (e.g., *[first]* for the value of the *first* variable), and the variable will be replaced by the value corresponding to each iteration of the loop. All aspects of stimuli presentation items can be defined using variables. If the widget used to define a property of a stimulus is not editable (e.g., a dropdown list), Visual DMDX provides a small orange button at the right to convert it into a text field in which you can create a name for the variable in brackets (see Fig. 3).

Other features: Export, choice of syntax, preview

In addition to the item editor, we built the following features into Visual DMDX. Researchers can choose to *export* their experiments not only to DMDX syntax but also to JSON (JavaScript Object Notation) format. This format is ideally suited for use by other web applications. Visual DMDX users can export their experiment to JSON and then import all or parts of it to a new Visual DMDX experiment. Thus, it is possible to store *offline backups* of Visual DMDX experiments, or even to generate JSON content with an auxiliary program written in Python or Java and then import it to Visual DMDX. Regarding the export to DMDX, it is possible to choose between traditional DMDX syntax and modern syntax (verbose). Finally, Visual DMDX provides researchers with a *preview mode* to check whether the order of the items and stimuli configured in the editor is correct or not.

Illustrative examples

The best way to illustrate the potential of Visual DMDX in setting up DMDX experiments is through examples. In the first example, we will create a *lexical decision experiment* in Visual DMDX similar to the one presented in the tutorial by Jiang (2012). In the second example, we use loops to repeat several items in an affective priming task with multimedia stimuli. Finally, we improve both examples using Visual DMDX's advanced features: variables and stimuli lists.

Lexical decision example

The first thing to do to create a new experiment in Visual DMDX is to access the http://visualdmdx.com website and click the "New Project" button. You will be redirected automatically to the editor of experiments. In the address bar of your browser you will see the unique identifier assigned to the project (e.g., http://visualdmdx.com/edit/4snvk2). It is highly recommended to save this web address, either by marking it as a favorite in the browser, using a social bookmarking platform (e.g., Delicious), or writing it down in suitable places.

The experiment editor includes a toolbar at the top where you can save your changes ("Save" button, located on the right), access previously saved versions of the experiment ("Versions" button), preview the experiment ("Preview" button), export the experiment to DMDX or JSON ("Export" button), import the experiment from JSON ("Import" button), or return to the editor from other options ("Edit" button). Below the toolbar there is a blank text field where one can define a description of the experiment. This description will not be exported to DMDX syntax, but allows one to add explanatory information to the project in Visual DMDX. In this example, type "Lexical decision example" into the text field.

Subsequently, we start defining the general parameters of the experiment. In Visual DMDX the parameters are divided into three groups: input, output, and other. In the "Input" group there are options related to the input of the participant. Here, we set "keyboard" as the input device (default) and we use 3,000 ms as the default time limit for a participant's responses to 3,000 ms. In the "Output" group, we set up a video mode supported by the graphics card of the computer that will run the experiment and define other parameters such as feedback (no feedback, outcome, or outcome and response time), frame duration, continuous run (whether the following item is shown automatically or not), delay

between items, font size, font color, and background color. In the "Other" group, one sets parameters related to the experiment's data logging procedure. After setting all the parameters, click the "Save" button. Visual DMDX then saves the first version of the experiment (accessible from http://visualdmdx.com/edit/4snvk2/1).

To add items (i.e., trials) to your experiment, click the "Items" button on the left side of the editor. After you are in the items editor, you see three buttons at the top, one for each item type. Press the "New instructions" button and a new light blue item box is added. Click on the item and its details are shown. Here you can define a description (it will not be exported to DMDX) and an instructions text that will be displayed to the participant. Write the following text: "This is a lexical decision experiment. Press YES if a letter string is a word. Press NO if it is not a word. Respond as quickly and as accurately as possible. Now press spacebar to continue." After an item has been defined, you can click its header to hide the details.

Now add a new stimuli presentation item. To do this, click on the "New item" button and a new dark blue item box will be added. Then click on the header of this item to display details. Describe the item using the description field, to make it easier to identify this item when more items are added. In this experiment, trials consist of a fixation point displayed for 30 ticks, followed by a word that is displayed for 60 ticks. At the beginning of this second stimulus you want DMDX to start recording the participant's response time. To do this, click the "Text" button next to the words "Add stimuli." After the stimulus has been added, click on its header and set the text to show the fixation point ("*"). Leave the rest of the stimulus properties on default values and reclick the header of the stimulus presentation item to hide its details. To add the word for this trial in the lexical decision task, add another "Text" stimulus and set the text to the desired word (e.g., "Permit"). Also, set the presentation time of this stimulus to 60 ticks and leave the rest of its properties with default values. Finally, it is important to set the time when DMDX starts to record the participant's response, using the dropdown list next to the words "Clock on." It is also necessary to define what response is expected for this item, using the dropdown list next to the word "Response." You have finished the stimuli presentation item definition process (see Fig. 5). To avoid losing your work, click the "Save" button again. (This version is now accessible from http://visualdmdx.com/edit/4snvk2/2.)

Fig. 5 Variable editor

Since the other trials of this experiment are very similar to the one above, we will use the cloning tool to save time when creating the remaining items. There is a button with an icon of a wrench at the left side of the header of each item. When you click this button, a modal dialog box with two options will be shown: Delete and Clone. Add a copy of the current item to the experiment by clicking the "Clone" button. Perform this action as many times as necessary to create all trials in your experiment (a total of six items, in this example). After all the cloned items have been created, modify them to change the words and the expected response (negative or positive). To avoid losing your work, click the "Save" button again. (This version is now accessible from http://visualdmdx.com/edit/4snvk2/3.)

Finish the editing process by adding another instructions item with this text: "Thank you for your participation. End of experiment." Once we have fully defined the experiment, we can export it to DMDX syntax. To make sure everything is correct, click the "Preview" button in the top toolbar and, once there, click the "Generate preview" button. The preview mode provides a simplified version of what DMDX will display. This animation does not take into account the colors or sizes of stimuli, there is no chance to respond to stimuli, and instruction items are shown only briefly (it is not necessary to press the spacebar). To reposition an item, return to the items editor, click on the item, and drag it to the desired position. Once everything is as expected, click the "Save" button again. (This version is now accessible from http://visualdmdx.com/edit/4snvk2/4.)

Access the "Export" option from the top toolbar. From here, export the experiment to DMDX syntax by clicking on the "Export to DMDX" button (see Table 2) or save a backup of your work using the "Export to JSON" option. This format allows you to create a new project in Visual DMDX and import to it all settings from a previous experiment using the "Import" option from the top toolbar. Therefore, creating modified versions of the same experiment is straightforward.

Affective priming example

Since the seminal article on affective priming by Fazio, Sanbonmatsu, Powell, and Kardes appeared in 1986, much research has been conducted on the automaticity of evaluation (for reviews, see Fazio, 2001; Klauer & Musch, 2003; Wentura & Rothermund, 2007).

In this example we will show how to use Visual DMDX's advanced features, such as multimedia stimuli or loop items with scrambling. To create this experiment, access the Visual DMDX website and click the "New experiment" button. After you are in the editor, define the description of the

Table 2 Lexical decision example DMDX item file generated by Visual DMDX

```
<ep><t 3000><d 5><dfd 30><id keyboard><vm 640, 480, 480, 8, 0><nfb><dwc 255255255><dbc 000000000><dfs 12><azk><rcot>
<! ———————————————————————————————————————————————————————————————————————— >
<! Experiment: Lexical decision example>
<! Created using Visual DMDX: http://visualdmdx.com/edit/4snvk2 >
<! ———————————————————————————————————————————————————————————————————————— >
<eop>
00 “This is a lexical decision experiment. Press YES if a letter string is a word. Press NO if it is not a word. Respond as quickly and accurately as you can. Now press spacebar to continue.”;
+2 “*” / * <% 60> “Permit” / ;
-3 “*” / * <% 60> “Dital” / ;
-4 “*” / * <% 60> “Faity” / ;
+5 “*” / * <% 60> “Replay” / ;
-6 “*” / * <% 60> “Larive” / ;
+7 “*” / * <% 60> “Heavy” / ;
00 “End of experiment. Thanks.”;
```

experiment and set the general parameters, as discussed in the previous example. Then, create an instructions item explaining to participants how to proceed: “Welcome to this experiment. Press LIKE key or DISLIKE key to indicate your feelings about each word. Respond as quickly and as accurately as possible. Now press spacebar to continue.” Save the state of the experiment by pressing the “Save” button. (This version is now accessible from http://visualdmdx.com/edit/jahjq5/1.)

The affective priming example consists of an unordered list of 40 items. In this list there are four different types of primes: (1) text, (2) image, (3) sound, and (4) video. Within each type, there will be five items, which will be shown two times each throughout the experiment (5 items × 2 times × 4 types = 40 items). First, create a stimuli presentation item with the following stimuli: (1) a text stimulus for the initial fixation point (text: “+,” duration: 30 ticks), (2) a blank stimulus of 30 ticks of duration, (3) a text stimulus to indicate the prime word (text: prime word, duration: 18 ticks), (4) a 12-ticks blank stimulus, and (5) a text stimulus to show the word that must be assessed by the participant (text: target word, duration: 60 ticks). Then set the clock to start running at the beginning of the 5th stimulus and set the expected response (positive/negative). Click the “Save” button; this version is now accessible from http://visualdmdx.com/edit/jahjq5/2.

After creating the stimuli presentation item, clone it five times and adapt the content of the clones to create the items corresponding with trials using primes of type text (the list of primes and targets used in this example is shown in Table 3). Because each of these items should be presented twice, add a loop item and place it between the initial instructions item and the first stimuli presentation item (drag and drop). Then, set the number of iterations (2) and the last item in the loop (item number 7). Save the state of the experiment by pressing the “Save” button. (This version is now accessible from http://visualdmdx.com/edit/jahjq5/3.)

To do the same with other types of primes, clone the last of the stimuli presentation items and show its details. Then, press the button to add a new stimulus (BMP/JPG, WAV, Video) and add it to the end of the stimuli list. Configure the path to the media file and set its duration to 18 ticks. Delete the text stimulus in the third place in the stimuli list by pressing the “X” button on its header. Then put the new stimulus in that position (drag and drop). For the rest of the items, repeat the actions explained in the previous paragraph: (1) clone this item five times, (2) adapt the content, and (3) create a new loop item and place it at the beginning of this set of five items. Perform the same procedure with the items that show audio and video primes. Finally, click the “Save” button again. (This version is now accessible from http://visualdmdx.com/edit/jahjq5/4.)

Now you have created an array of 40 items (four blocks of five items to be repeated two times). To scramble it, simply create a new loop item, place it at the very beginning (immediately after the initial instructions item), set the last item as the end of the loop, define one iteration, and enable the scramble option. Note that you could also have enabled the scrambling option in each of the four existing loop items to randomize trials within blocks. The result would not be the same: If you create a new loop item that scrambles the whole list of items, all items can end up anywhere in the 40-item list, but if you enable scrambling in each of the four loops of each type of prime (i.e., text, image, audio, and video), the first 10 items will be of type text, the following 10 of type image, the next 10 of type audio, and the last 10 of type video. It is important to realize the implications of the chosen option in terms of randomization of all items versus randomization within blocks.

Table 3 Affective priming example DMDX item file generated by Visual DMDX

```
<ep><t 1000><d 5><dfd 30><id keyboard><vm 640, 480, 480, 8, 0><nfb><dwc 255255255><dbc 000000000><dfs 12><azk><rcot>
<! ———————————————————————————————————————————————————————————————————— >
<! Experiment: Affective priming example>
<! Created using Visual DMDX: http://visualdmdx.com/edit/jahjq5 >
<! ———————————————————————————————————————————————————————————————————— >
<eop>
00 “Welcome to this experiment. Press LIKE key or DISLIKE key to indicate your feelings about each word. Respond as quickly and as accurately as possible. Now press spacebar to continue.”;
+2 “+” / /<% 18> “WAR” / <% 12> / * <% 60> “FUN” / ;
-3 “+” / /<% 18> “FLOWER” / <% 12> / * <% 60> “PAIN” / ;
+4 “+” / /<% 18> “WAR” / <% 12> / * <% 60> “FUN” / ;
-5 “+” / /<% 18> “FLOWER” / <% 12> / * <% 60> “PAIN” / ;
+6 “+” / /<% 18> “SUGAR” / <% 12> / * <% 60> “BEAUTY” / ;
-7 “+” / /<% 18> “EVIL” / <% 12> / * <% 60> “KILL” / ;
+8 *+“ ” / /<% 18> “CANDY” / <% 12> / <% 60> “FLOWER” / ;
+9 “+” / /<% 18><bmp> “war-image” / <% 12> / * <% 60> “FUN” / ;
+10 “+” / /<% 18><bmp> “candy-image” / <% 12> / * <% 60> “FLOWER” / ;
-11 “+” / /<% 18><bmp> “flower-image” / <% 12> / * <% 60> “PAIN” / ;
+12 “+” / /<% 18><bmp> “sugar-image” / <% 12> / * <% 60> “BEAUTY” / ;
-13 “+” / /<% 18><bmp> “evil-image” / <% 12> / * <% 60> “KILL” / ;
+14 “+” / /<% 18><bmp> “war-image” / <% 12> / * <% 60> “FUN” / ;
+15 “+” / /<% 18><bmp> “candy-image” / <% 12> / * <% 60> “FLOWER” / ;
-16 “+” / /<% 18><bmp> “flower-image” / <% 12> / * <% 60> “PAIN” / ;
+17 “+” / /<% 18><bmp> “sugar-image” / <% 12> / * <% 60> “BEAUTY” / ;
-18 “+” / /<% 18><bmp> “evil-image” / <% 12> / * <% 60> “KILL” / ;
+19 “+” / /<% 0><svp start><wav 2> “war-audio” / <% 12> / * <% 60> “FUN” / ;
+20 “+” / /<% 0><svp start><wav 2> “candy-audio” / <% 12> / * <% 60> “FLOWER” / ;
-21 “+” / /<% 0><svp start><wav 2> “flower-audio” / <% 12> / * <% 60> “PAIN” / ;
+22 “+” / /<% 0><svp start><wav 2> “sugar-audio” / <% 12> / * <% 60> “BEAUTY” / ;
-23 “+” / /<% 0><svp start><wav 2> “evil-audio” / <% 12> / * <% 60> “KILL” / ;
+24 “+” / /<% 0><svp start><wav 2> “war-audio” / <% 12> / * <% 60> “FUN” / ;
+25 “+” / /<% 0><svp start><wav 2> “candy-audio” / <% 12> / * <% 60> “FLOWER” / ;
-26 “+” / /<% 0><svp start><wav 2> “flower-audio” / <% 12> / * <% 60> “PAIN” / ;
+27 “+” / /<% 0><svp start><wav 2> “sugar-audio” / <% 12> / * <% 60> “BEAUTY” / ;
-28 “+” / /<% 0><svp start><wav 2> “evil-audio” / <% 12> / * <% 60> “KILL” / ;
+29 “+” / /<% 12><dv> “war-video” / <% 12> / * <% 60> “FUN” / ;
+30 “+” / /<% 12><dv> “candy-video” / <% 12> / * <% 60> “FLOWER” / ;
-31 “+” / /<% 12><dv> “flower-video” / <% 12> / * <% 60> “PAIN” / ;
+32 “+” / /<% 12><dv> “sugar-video” / <% 12> / * <% 60> “BEAUTY” / ;
-33 “+” / /<% 12><dv> “evil-video” / <% 12> / * <% 60> “KILL” / ;
+34 “+” / /<% 12><dv> “war-video” / <% 12> / * <% 60> “FUN” / ;
+35 “+” / /<% 12><dv> “candy-video” / <% 12> / * <% 60> “FLOWER” / ;
-36 “+” / /<% 12><dv> “flower-video” / <% 12> / * <% 60> “PAIN” / ;
+37 “+” / /<% 12><dv> “sugar-video” / <% 12> / * <% 60> “BEAUTY” / ;
-38 “+” / /<% 12><dv> “evil-video” / <% 12> / * <% 60> “KILL” / ;
00 “Thank you for your collaboration”;
```

Finally, add a new instructions item with the text “Thank you for your collaboration” and save the experiment one more time. (This version is now accessible from http://visualdmdx.com/edit/jahjq5/5.)

Table 4 Lexical decision example using variables generated by Visual DMDX

```
<ep><t 3000><d 5><dfd 30><id keyboard><vm 640, 480, 480, 8, 0><nfb>
<dwc 255255255><dbc 000000000><dfs 12><azk><rcot>
<! ———————————————————— >
<! Experiment: Lexical decision example with variables>
<! Created using Visual DMDX: http://visualdmdx.com/edit/s1zsyd >
<! ———————————————————— >
<eop>
00 <ln -3> “This is a lexical decision experiment. ”,
<ln -2> “”,
<ln -1> “Press YES if a letter string is a word. ”,
<ln 0> “Press NO if it is not a word. ”,
<ln 1> “Respond as quickly and accurately as you can. ”,
<ln 2> “”,
<ln 3> “Now press spacebar to continue.”;
-2 “*” / * <% 60> “PUMBLEAN” / ;
+3 “*” / * <% 60> “FATHER” / ;
+4 “*” / * <% 60> “MAINTAIN” / ;
-5 “*” / * <% 60> “SAPKUD” / ;
-6 “*” / * <% 60> “GLUATION” / ;
-7 “*” / * <% 60> “PROBLE” / ;
+8 “*” / * <% 60> “DECADE” / ;
+9 “*” / * <% 60> “PASTRY” / ;
+10 “*” / * <% 60> “MANAGER” / ;
+11 “*” / * <% 60> “AIRCRAFT” / ;
+12 “*” / * <% 60> “MAJORITY” / ;
-13 “*” / * <% 60> “BOWBLE” / ;
-14 “*” / * <% 60> “HINERNOT” / ;
+15 “*” / * <% 60> “IDENTITY” / ;
-16 “*” / * <% 60> “SHILDIEN” / ;
+17 “*” / * <% 60> “CELLAR” / ;
+18 “*” / * <% 60> “CULTURE” / ;
+19 “*” / * <% 60> “KITTEN” / ;
-20 “*” / * <% 60> “BLICT” / ;
+21 “*” / * <% 60> “CHARGE” / ;
-22 “*” / * <% 60> “BOLECHUP” / ;
-23 “*” / * <% 60> “RENSOR” / ;
+24 “*” / * <% 60> “MILITARY” / ;
+25 “*” / * <% 60> “DEVICE” / ;
-26 “*” / * <% 60> “SHUSTER” / ;
-27 “*” / * <% 60> “GLOIN” / ;
-28 “*” / * <% 60> “JINNER” / ;
-29 “*” / * <% 60> “GROSHARY” / ;
-30 “*” / * <% 60> “CAPURALT” / ;
-31 “*” / * <% 60> “CORTH” / ;
+32 “*” / * <% 60> “PASSAGE” / ;
+33 “*” / * <% 60> “DRYER” / ;
-34 “*” / * <% 60> “AUBLE” / ;
+35 “*” / * <% 60> “PEPPER” / ;
-36 “*” / * <% 60> “BRANCE” / ;
+37 “*” / * <% 60> “ELECTION” / ;
-38 “*” / * <% 60> “BISTLE” / ;
+39 “*” / * <% 60> “PLASTIC” / ;
-40 “*” / * <% 60> “CHERP” / ;
+41 “*” / * <% 60> “HANDLE” / ;
00 “End of experiment. Thanks.”;
```

Evoke the Preview option and check that everything is correct; then export the experiment to DMDX syntax. As mentioned before, each time you click the “Export to DMDX” button, a different order of items is generated. You can press this button as many times as needed if you want to prepare several versions of the experiment with a random but known ordering for each participant.[1] Table 3 shows the resulting DMDX syntax for this example.

Scaling up our examples

The previous two examples demonstrated that Visual DMDX can be used to create DMDX items files without knowing DMDX syntax. However, we used Visual DMDX as a mere translator of values in web forms to DMDX keywords. The practical usefulness of Visual DMDX can be dramatically improved by way of advanced features such as the use of variables in loops or the ability to import lists of stimuli.

In the previous lexical decision example, we have seen how easy it is to create an experiment with similar items by cloning them. However, this approach does not scale well for a large number of items. Therefore, for experiments with many items, we recommend using an advanced feature of Visual DMDX: loop items’ variables.

Suppose you want to create a lexical decision experiment with 40 items, using variables. First, add four items to the experiment: (1) an instructions item at the beginning, (2) a loop item, (3) a stimuli presentation item, and (4) another instructions item to indicate the end of the experiment. Second, configure the stimuli presentation item in the same way that you prepared it in the previous version of the lexical decision example. Third, set to 40 the number of iterations in the loop item and select the stimuli presentation item in the dropdown list where the scope of the loop is defined (i.e., the “To” property of the loop item).

If you were to conclude here, you would have created an experiment with 40 identical items. It is at this moment when variables come into play. To use the variables editor,

[1] Alternatively, if you want DMDX to scramble the items for each participant dynamically, you have to add previously explained scrambling keywords manually (i.e., Scramble, ScrambleSeed, Grouping).

Table 5 List of stimuli in CSV prepared to be imported by Visual DMDX

```
text,+,30,true,center,middle
blank,30
text,[prime],18,true,center,middle
blank,12
text,[target],60,true,center,middle
```

press the "Set variables" button of the loop item. After you are in the variables editor (see Fig. 5), press the "Add variable" button and then define the name of the variable and what value it will take for each of the iterations of the loop. Set the variable name to "word" and add words and nonwords for a total of 40. When you are done, click the "Add variable" button again to add a variable called "response," and then set the expected response (negative or positive) for each iteration of the loop. Considering that editing the list of values can be tedious, you can press the "Import variables" button instead, and a simple form will be provided to paste a list of variables and values in CSV format and import them to the variable list using the "Import" button.

Finally, you need to edit the item presentation of stimuli to make use of the newly created variables. Click the orange button with a loop icon next to the dropdown list of the Response property of the stimuli presentation item. After you click this button, the dropdown list becomes a text field. Write "[response]" (without the quotes) into the text field. Then add two text stimuli, one for the fixation point (a "*," in our example) and another one for the word that will be evaluated by participants. To change the word in each trial, write "[word]" (without the quotes) as the Text property of the second stimulus. Now save the changes. (A version of this example is available in http://visualdmdx.com/edit/s1zsyd.) Check that everything went well by pressing the "Export to DMDX" button in the "Export" tab of Visual DMDX (see Table 4).

The example of affective priming would benefit not only from the use of variables in loop items, but also from the option to import whole lists of stimuli. Preparing a complex item can become an easy task if the experimenter defines a list of its stimuli in CSV format and uses the "Import stimuli" button. It is necessary that each row of the stimuli list begins with the type of stimulus that is to be added (i.e., blank, text, bmp, jpg, wav, video) and contains an ordered list of values for each of the properties of the item (see Table 5).

Discussion

In this article we presented the Visual DMDX web application, an authoring tool for creating item files for DMDX. Our goal is to facilitate experiment creation with the widely used DMDX syntax, in particular to ease the first steps for novice users of DMDX. Therefore, the core functionality provided by DMDX is available in Visual DMDX, and experiments using DMDX can now be created without knowledge of DMDX syntax. As a further improvement, Visual DMDX-specific features such as loop items provide a simple way to avoid the complexity of creating loops and shuffling items in DMDX. Newly added features that may also be attractive to experts of DMDX include multiple ways to export experiment code, easy backup, and a preview mode for experiments.

We will continue adding more routines. Improving the preview mode and the inclusion of new functionalities of DMDX (such as button mapping or conditional branching) are among our future plans for Visual DMDX. The export/import function to and from JSON may serve as an interface and future API to bridge the grand and powerful DMDX tradition to the modern world of web-based experimentation.

Visual DMDX is published under an open-source license (Affero GPL, version 3.0; see Free Software Foundation, 2007). Therefore, any researcher is free to upgrade and expand its features.

Author note We thank Julen Telleria for assistance in coding. Support for this research was provided by Grant PSI2011-26965 from Dirección General de Investigación of the Spanish Government and Grant IT363-10 from the Basque Government. The authors declare that there was no conflict of interest in the publication of this study.

References

Birnbaum, M. H. (2000). SurveyWiz and FactorWiz: JavaScript Web pages that make HTML forms for research on the Internet. *Behavior Research Methods, Instruments, & Computers, 32,* 339–346.

Fazio, R. H. (2001). On the automatic activation of associated evaluations: An overview. *Cognition and Emotion, 15,* 115–141.

Fazio R. H., Russell H., David M. Sanbonmatsu, Martha C. Powell, and Frank R. Kardes (1986). On the Automatic Activation of Attitudes. *Journal of Personality and Social Psychology 50*, 229–238.

Forster, K. I., & Forster, J. C. (2003). DMDX: A Windows display program with millisecond accuracy. *Behavior Research Methods, Instruments, & Computers, 35,* 116–124.

Free Software Foundation. (2007). *GNU Affero General Public License, version 3.0*. Retrieved December 18, 2013 from: http://www.gnu.org/licenses/agpl-3.0.html

Freelon, D. (2010). ReCal: Intercoder reliability calculation as a Web service. *International Journal of Internet Science, 5,* 20–33.

Freelon, D. (2013). ReCal OIR: Ordinal, interval, and ratio intercoder reliability as a web service. *International Journal of Internet Science, 8,* 10–16.

Garaizar, P., & Reips, U.-D. (2013). Build your own social network laboratory with Social Lab: A tool for research in social media.

Behavior Research Methods. doi:10.3758/s13428-013-0385-3. Advance online publication.

Göritz, A. S., & Birnbaum, M. H. (2005). Generic HTML Form Processor: A versatile PHP script to save Web-collected data into a MySQL database. *Behavior Research Methods, 37,* 703–710.

Jiang, N. (2012). *Conducting reaction time research in second language studies*. New York: Routledge.

Klauer, K. C., & Musch, J. (2003). Affective priming: Findings and theories. In J. Musch & K. C. Klauer (eds.), *The psychology of evaluation: Affective processes in cognition and emotion* (pp. 7–50). Mahwah, NJ: Erlbaum.

Krantz, J. H. (2000). Tell me, what did you see? The stimulus on computers. *Behavior Research Methods, Instruments, & Computers, 32,* 221–229.

Mathôt, S., Schreij, D., & Theeuwes, J. (2012). OpenSesame: An open-source, graphical experiment builder for the social sciences. *Behavior Research Methods, 44,* 314–324.

Peirce, J. W. (2007). PsychoPy: Psychophysics software in Python. *Journal of Neuroscience Methods, 162,* 8–13.

Reips, U.-D. (2001). The Web Experimental Psychology Lab: Five years of data collection on the Internet. *Behavior Research Methods, Instruments, & Computers, 33,* 201–211.

Reips, U.-D. (2010). Design and formatting in Internet-based research. In S. Gosling & J. Johnson (Eds.), *Advanced methods for conducting online behavioral research* (pp. 29–43). Washington, DC: American Psychological Association.

Reips, U.-D., & Funke, F. (2008). Interval level measurement with visual analogue scales in Internet-based research: VAS Generator. *Behavior Research Methods, 40,* 699–704.

Reips, U.-D., & Garaizar, P. (2011). Mining Twitter: Microblogging as a source for psychological wisdom of the crowds. *Behavior Research Methods, 43,* 635–642.

Reips, U.-D., & Lengler, R. (2005). The *Web Experiment List*: A Web service for the recruitment of participants and archiving of Internet-based experiments. *Behavior Research Methods, 37,* 287–292.

Reips, U.-D., & McClelland, G. (2013). Interactive applets on the Web for methods and statistics. *University of Amsterdam, AIAS Working Paper 132*.

Reips, U.-D., & Neuhaus, C. (2002). WEXTOR: A Web-based tool for generating and visualizing experimental designs and procedures. *Behavior Research Methods, Instruments, & Computers, 34,* 234–240.

Schneider, W., Eschman, A., & Zuccolotto, A. (2002). *E-Prime user's guide*. Pittsburgh, PA: Psychology Software Tools.

Stieger, S., & Reips, U.-D. (2008). Dynamic Interviewing Program (DIP): Automatic online interviews via the instant messenger ICQ. *CyberPsychology and Behavior, 11,* 201–207.

Stieger, S., & Reips, U.-D. (2010). What are participants doing while filling in an online questionnaire: A paradata collection tool and an empirical study. *Computers in Human Behavior, 26,* 1488–1495.

Wentura, D., & Rothermund, K. (2007). Paradigms we live by: A plea for more basic research on the IAT. In B. Wittenbrink & N. Schwarz (eds.), Implicit measures of attitudes (pp. 195–215). New York: Guilford Press.