Dynamic online surveys and experiments with the free open-source software *dynQuest*

JENS D. M. RADEMACHER

Centre for Mathematics and Computer Science (CWI, MAS), Amsterdam, The Netherlands

AND

SONIA LIPPKE

Freie Universität Berlin, Germany

With computers and the World Wide Web widely available, collecting data through Web browsers is an attractive method utilized by the social sciences. In this article, conducting PC- and Web-based trials with the software package *dynQuest* is described. The software manages dynamic questionnaire-based trials over the Internet or on single computers, possibly as randomized control trials (RCT), if two or more groups are involved. The choice of follow-up questions can depend on previous responses, as needed for matched interventions. Data are collected in a simple text-based database that can be imported easily into other programs for postprocessing and statistical analysis. The software consists of platform-independent scripts written in the programming language PERL that use the common gateway interface between Web browser and server for submission of data through HTML forms. Advantages of *dynQuest* are parsimony, simplicity in use and installation, transparency, and reliability. The program is available as open-source freeware from the authors.

Section 1 Introduction

In recent years, the Internet has been increasingly used to collect behavioral data in many areas of psychology and the social sciences (see, e.g., Bailey, Foote, & Throckmorton, 2000; Birnbaum, 1999; Buchanan & Smith, 1999; Krantz, Ballard, & Scher, 1997). The ubiquity of personal computers and the World Wide Web (WWW) gives researchers the potential to (1) reach large samples (Birnbaum, 1999) of persons with diverse socioeconomic status and age, and from different geographic regions (Gosling, Vazire, Srivastava, & John, 2004); (2) match questions and page contents to the specific needs of the study participants (Nigg, 2003) (e.g., sections that are irrelevant to the current study participant can be skipped automatically); (3) conduct randomized control trials (RCT)—the gold standard for theory testing and intervention research (Westen, Novotny, & Thompson Brenner, 2004)—in which group assignment can be done automatically and invisibly to the participant; and (4) conduct computer aided telephone interviews (CATIs), computer aided interviews (CAIs), or self-administered questionnaires.

Researchers have developed a number of techniques for creating computer-assisted assessments and Web-based experiments that are comparable with traditional paper-and-pencil methods (Gosling et al., 2004) and that are summarized under the term *trial* in the following. The implementations of these frequently use commercial soft-

ware and plugins, such as Authorware and ShockWave (see, e.g., Williams, McGraw, & Tew, 1999; Wolfe, 1992), Java and JavaScript (see Baron & Siepmann, 2000), and common gateway interface (CGI) programs in various programming languages such as PERL and C11 (e.g., Morrow & McKee, 1998).

However, several problems may arise. On the one hand, participant-side browser plugins or applications, such as Java and JavaScript, are not necessarily available on the participant's computer (Wolfe & Reyna, 2002) and may require large computer memory and a long download time (cf. software used, e.g., by Mar, Chabal, Anderson, & Vore, 2003). Writing in a programming language is error prone. And buying a software package or paying a programmer is cost intensive.

On the other hand, when relying on HTML forms, which are relatively simple to create, examples of major problems are the automatic, secure storage of data and the dynamic, answer-dependent linking and adjustment of consecutive pages. Many consumers use the self-administered Internet to acquire information. However, the effectiveness of the information received depends on various psychological factors (Cummins et al., 2003), so that offers on the Internet should provide tailored information, which is not possible with a static page. It is more effective to ask key questions and, depending on the answers, provide subsequent information (Kirsch & Lewis, 2004). Likewise, individuals feel more appreciated and

more motivated to continue the trial in a questionnaire that only asks appropriate questions (e.g., upon asking about smoking, only those who gave a positive answer should be asked whether they want to quit).

However, although researchers may be able to create simple HTML pages and load them onto the Internet, they are often not familiar with programming sophisticated Web pages or complex expert system software packages (see Cummins et al., 2003; Evers et al., 2003; Kirsch & Lewis, 2004). A solution was needed to provide an interface for the dynamic linking of Web pages with data collection options that is readily available, easy to use, and reliable.

The software package *dynQuest* is designed for these purposes, and it has been used successfully in several surveys and experiments (see, e.g., Lippke, Sniehotta, & Luszczynska, 2005; Lippke, Ziegelmann, & Schwarzer, 2004). In the online study by Lippke et al. (2005), *dynQuest* managed the dynamic questionnaire and data storage for 812 individuals who visited the questionnaire, 34 of whom did not complete the entire questionnaire or answered less than 50% of the questions. Those who dropped out were not significantly different from those who remained in the study (see Lippke et al., 2005), and this dropout rate of 4.2% is smaller or comparable to other online studies (cf. Mar et al., 2003) and paper-and-pencil versions (see, e.g., Gosling et al., 2004; Schwarzer et al., 2006). In two other online studies with *dynQuest*, a two-measurement-point design was used. For an online study on physical activity, the initial sample consisted of 1,229 individuals, 881 of whom (68.9%) provided their e-mail addresses in order to receive an invitation for a follow-up assessment. As a result, 365 respondents (41.4% of those who potentially could have participated) answered the follow-up questionnaire (Schwarzer et al., 2006). In an online study on nutrition, the initial sample contained 1,905 individuals; 1,659 (87.1%) provided their e-mail addresses to receive an invitation for a follow-up assessment, and 700 (42.2% of those who potentially could have participated) took part in the follow-up questionnaire (Schwarzer et al., 2006). In another study using *dynQuest* for CAI and randomizing participants to one of three groups, 560 patients were followed up with a CATI that also used *dynQuest*. Forty-four patients at the first follow up and 56 patients at the second follow up could not be interviewed because they were unavailable (Lippke et al., 2004).

In conclusion, the software was used successfully to administer different data-collection modes for large samples. It was found to be very helpful by interviewers, and online-study participants expressed their satisfaction in an open-answer format. Practical advantages of using *dynQuest* (beyond the availability of the programmer in these cases) were the ease of installation, the reliability of the software, and the simple and transparent way it works, which allowed the ability to track and correct errors in trial programming and to repair the database, if necessary. Moreover, specific *dynQuest* commands could easily be added to HTML pages written by people without knowledge of the software.

In summary, *dynQuest* is focused on linking dynamic trial pages. Its use requires some technical deliberations concerning HTML and file management, but it is nevertheless simple, reliable, and transparent. It is freely available to researchers for computer-assisted and Web-based trials. Thus, the aim of the present article is to explain the use and functionality of *dynQuest* in detail.

Section 2 The Software

For user friendliness, *dynQuest* comes in two parallel versions, one for single PCs under Windows and the other for networked computers under UNIX or Linux, respectively (see Table 1). The basis of a trial for *dynQuest* is a flow diagram (see Figure 1) where the choice of the following page can depend on responses in the previous pages. The pages of the trial, which need to be in HTML format, may be created with any text or HTML editor (such as DreamWeaver or FrontPage).

The software runs in the background—invisible to the study participant—and manages the page flow and data storage. It analyzes the data submitted from the current page of the questionnaire, sends the appropriate follow-up page to the browser, and stores the new data in a database on the server.

The HTML pages contain text and forms for the questions and encode the algorithm for the choice of the next page as programmed by the user (similar to the software PHP). Hence, the flow diagram is not implemented as a whole, but each node of the diagram is an HTML page that contains the links only to adjacent follow-up pages. *dynQuest* applies quite universally to create a database from interviews via Web browsers and, in addition, follow links and manipulate pages, depending on the data submitted. It has also been proven useful for static questionnaires and for registering students or future study participants. Similar commercial software is available, but the advantages of *dynQuest* are that it is easy to use and install, simple, transparent, focused on the purpose, and it poses no restrictions on the layout of the trial pages. Furthermore, it is open source, free, platform independent, and it only uses standard capabilities of browsers (HTML forms and CGI); it does not use PHP, Java, JavaScript, cookies, or pop-up windows.

The software package. The programs in the *dynQuest* package are text files (scripts) written in the (freely available) programming language PERL. *dynQuest* makes use of the CGI between Web browsers and servers; hence, to use it on a single PC, an HTML browser, PERL, and an http server (freely available, e.g., Apache) would need to be installed (see Section 4 and the Appendix). Only basic HTML and operating system knowledge is required for running *dynQuest*. The main steps for making a dynamic questionnaire or intervention trial available on the Internet are as follows. (1) Check that the programming language PERL is available on your Web account and that study participants can run the *dynQuest* CGI scripts through a browser (see the Appendix), (2) create HTML pages for the nodes of the flow chart of your questionnaire using the *dyn-*

Table 1
Files of the Software Package for Use on PC or Unix/Linux As Indicated by the Name

File Name	Function
dynQuest.pl.cgi (one for PC and one for Unix)	Main script, managing the flow of pages and database entries. PC and Unix version have the same names so that the same HTML pages can be used for both.
definitions.txt	Text file that contains definitions of keywords and other <i>dynQuest</i> options; these may be changed by the user at will (Section 3.1.1).
showQuestUnix.pl.cgi, showQuestPC.pl.cgi	Analyze and display the link structure (flow chart) (Section 3.2.6).
cleanQuestUnix.pl.cgi, cleanQuestPC.pl.cgi	Remove redundant entries from the database (Section 3.1.7).

Quest commands for the linking structure (see Section 3), and (3) copy these pages and all relevant files (see Tables 1 and 2) from the *dynQuest* package into one folder.

The *dynQuest* files are open-source software that can be redistributed and/or modified under the terms of the GNU General Public License as published by the Free Software Foundation. The original copyright of the files rests with the first author. The parts of *dynQuest* are listed in Table 1.

To run a specific trial, further files beyond those containing the questions are needed, or may be helpful (see Table 2).

Users can open and view all of the files that are listed in Tables 1 and 2 in a text editor. We recommend not changing the CGI scripts. If there are questions or comments, please contact the authors. The software is under ongoing development, and further functions may be added in the future; see the Web page userpage.fu-berlin.de/~slippke/.

Limitations of the software. The layout of the trial needs to be programmed and checked by the user (advanced HTML editors can do this). Correct programming of *dynQuest* commands and spelling of item names is not assisted, and there is no help or software documentation

Figure 1. The multistage model algorithm and rules for stage assignment (adapted from Lippke, Sniechotta, & Luszczynska, 2005).

Table 2
Required and Optional Files Not Contained in the *dynQuest* Package

File Name	Function	Required or Optional
fieldnames.txt	Lists all fields, i.e., variables, names (Section 3.1.3)	Required. If not in folder, no data will be saved.
defaultDataFile.txt	"Data Base"; contains data status of all submissions	Optional. In case it is not in the folder, it is newly created (Section 3.1.7).
showPageNames.txt	Causes current file name to be displayed (Section 3.2.6)	Optional. Helpful for piloting and checking the trial prior to the actual data collection.
showLastForm.txt	Causes entries on prior page to be displayed (Section 3.2.6)	Optional. Helpful for piloting and checking the trial prior to the actual data collection.
showMsg.txt	Causes display of the processing of history dependent links.	Optional. Helpful for piloting and checking the trial prior to the actual data collection.

beyond this article. There is no automatic management of database files other than backups when purging the database (see Sections 3.1.7 and 4) (file management software can do this). The algorithm for choosing follow-up pages is somewhat limited concerning possible logical operations (see Section 3.2). *dynQuest* cannot check whether a participant's codeword is already in use by another participant, and the Internet communication channels are not secured against third-party interception or manipulation (which can be managed with other software).

As indicated, most of these aspects can be overcome in conjunction with other software. In particular, we expect combinations with the HTML preprocessor PHP to be fruitful. However, a discussion of details would go beyond the scope of this article.

Installation. *dynQuest* itself does not require any installation. Simply copy all relevant files into the folder of the trial pages. Note that the folders with scripts and HTML pages need to be accessible by the Web browser of study participants (see Section 4 and the Appendix).

Section 3 Programming the Trial

Relevant elements of a trial are (1) Page names of the trial pages—that is, the nodes in the flow chart (e.g., Figure 1) of the trial, (2) links and linking condition between pages—that is, the edges of the flow chart (e.g., Figure 1), (3) field names—that is, the names of items that participants can respond to, and (4) item values to be stored in the database as a response for a question with predefined possible responses (e.g., degree of agreement with a statement, range 1–5).

Each node in the flow chart of the trial is encoded in an HTML form (within an HTML file) containing the items, together with all outgoing edges—that is, the linking in-

formation to all follow-up nodes—invisible to the participant (see Table 3). Each item of the questionnaire requires a unique field name that is used as the name of the HTML form elements concerning this item, and as the column header in the database. For instance, items that inquire about the degree of agreement with statements often come in groups and may be arranged row-wise in a table whose columns contain fields that the participant can select. In HTML, such a selection can be realized—for example, via so-called radio buttons that can be activated by the participant; however, we do not discuss general HTML programming in this article.

The HTML form of a node must submit data to *dynQuest* via the so-called "post" method. Typically, the HTML code (located anywhere within the HTML body) is:

```
<form method="POST"
  action="dynQuest.pl.cgi">
  ...
</form>
```

Upon submission, the form data will be stored in the database, which is a text file generated automatically according to a given list of all field names (see Table 2). This database can be imported easily for postprocessing and (statistical) analysis into database or statistics software, such as SPSS or Excel (see Section 4.3).

After storing data, the follow-up page will be sent to the participant's browser by *dynQuest* according to the encoded linking, which allows for answer-dependent, static (answer-independent), or randomized links and manipulation of page contents. The correspondence between trial information and HTML form data is as follows.

The flow chart edges (see Figure 1 and the example below) are encoded either in hidden fields or in the names

Table 3
Correspondence of Trial and HTML Form Data

Trial	HTML Form
Item name	Input field name "<input [...] name=[...]>"
Item value, if predefined	Input field value "<input [...] value=[...]>". Typically several such form elements correspond to one item with different values, e.g., a set of radio buttons.
Item that is invisible to the participant, e.g., indicator of current page	Hidden field "<input type='hidden' [...]>"

and values of items. For instance, the form element name “next_sport” with value “page3a_1” by default means that when activating this form element (which might be a radio button for “volleyball”), the page “page3a.htm” will be loaded next, and the value “1” will be written into the database in the column “sport”. The HTML code for this and another radio button for “soccer” with value “2”, linking to another file could be:

```
<form method="POST"
action="dynQuest.pl.cgi">
Your favorite sport:<br>
<input type="radio" name="next_
sport" value="page3a_1">
Volleyball<br>
<input type="radio" name="next_
sport" value="page3b_2">
Soccer<br>
<input type="submit"
name="submission" value="send
data">
</form>
```

(The keyword “next”, the suffix “htm”, and the separator symbol “_” can be changed at will in the file “definitions.txt”; see Section 3.1.1.)

dynQuest makes frequent use of so-called “hidden” fields (“<input type=“hidden””), which are not shown in the browser, but whose values are submitted. These fields can be inserted anywhere within the HTML form that submits data to *dynQuest*. The only exception is several competing links on one page (see Sections 3.1.1 and 3.2.5).

We emphasize that on the first page of a trial—among other possible questions—an individual code is defined automatically or by the participant (see Section 3.1.2) that is used later to purge the database of multiple submissions. In addition, it allows the identification of participants in a longitudinal study. In the following sections, details of the HTML programming for *dynQuest* and its functionality are discussed.

Section 3.1

The Single HTML Page

In this section, we discuss all elements concerning a single HTML page; the linking of pages will be described in Section 3.2. The following subsections are relatively independent and do not need to be read in order. These subsections are (1) defaults and conventions, (2) initial page and code definition, (3) list of field names, (4) required entries, (5) dynamic page manipulation, (6) data status, (7) the database, (8) error handling, and (9) numerical functions.

Section 3.1.1

Defaults and Conventions

The file “definitions.txt” contains all keywords and symbols for *dynQuest* programs that may be changed by the user. These are listed line-wise with a brief explanation

separated from the keyword/symbol itself by an ASCII tabulator. Note that this file contains settings for both a single PC and network (UNIX) settings. Throughout this article, we refer to the default settings.

Since certain symbols are meaningful for *dynQuest*, some care has to be taken in naming files and items. Keywords of the definitions file cannot be used as item or file names, but the use of the separator symbol is somewhat flexible and is described in detail in the respective sections. The HTML suffix set in the definitions file will be added whenever it is not contained in a file name received from a form.

Since text fields in HTML forms have a default value, *dynQuest* allows the definition of a “nonvalue,” by default the minus sign “-”. If this value is encountered, the item is considered unanswered.

Further conventions and assumptions about the HTML forms are as follows. (1) If several form elements with the same name are submitted, the first of these in the order of submission (by standard, this is the order within HTML file) will be used; (2) if several simple conditional or unconditional links are submitted, the last of these will be used (see Section 3.2.5); (3) if semicolons and line breaks/carriage returns are encountered within submitted data, they will be replaced by commas or “|”, since they interfere with the database structure (see Section 3.1.7); and (4) *showQuest* (see Section 3.2.6) assumes that an input form element is contained entirely in one line of the HTML file; for example, there is no line break between “<input” and “name=” for the same input element. It also assumes that between two input fields, there is a line break in the HTML file.

Section 3.1.2

Initial Page and Code Definition

As mentioned above, the first page of the trial is used (possibly together with other questions) to determine the code for each participant, and thereby to identify the entries in the database that belong to the same participant. This code—for example, a selection of letters from first and last name and birthplace—is used to control multiple submissions and purge the database (see Section 3.1.7). The date and time of the first data delivery are the code until submission of a field for code definition, indicated by the keyword for code definition (by default “CODEDef”), that is, “CODEDefx”, where “x” is not relevant for *dynQuest*, but should be different to distinguish fields. For instance, two code words can be inquired by the HTML code (somewhere within the HTML form):

```
<input type="text" name="CODEDef1"
value="-">
<input type="text" name="CODEDef2"
value="-">
```

The values of all these fields are concatenated in the order of the form; for example, “Boston”, “Paula”, becomes “BostonPaula”, and is written into the database under the item “dqcode”. Usually code-defining fields are required fields (see Section 3.1.4).

Section 3.1.3

List of Field Names

The only information about the trial that is not contained in the HTML pages themselves (and the settings file “definitions.txt”) is a complete list of all field names that occur in the entire trial. This list needs to be provided by the user (via any text editor or generated by *showQuest*; see Section 3.2.6) in the form of a text file with line breaks between field names—for example:

```
fieldname1
fieldname2
...
```

The spelling has to match precisely the one in the HTML pages: Beware of blanks, lowercase versus capital letters, and so on. The ordering of these names is also used in the database, but it need not match the ordering in the trial. Since this file is read after each submission, it can be manipulated during a trial, for instance, to add items.

Section 3.1.4

Required Entries

Answers can be “required” in the sense that the page named (by default) “missingRequired.htm” is loaded when a required field has not received data. Study participants are thus forced to answer all such questions to get to the following trial page. The content of this page should encourage the participant to go back (by the “back” action/button of the browser) and complete the answers. However, we caution the user that study participants may get frustrated and drop out.

The requirement is implemented by a hidden field with the name “require” (by default), whose value is a list of field names that are required with the separator symbol in between (by default, “_”); for example, at an arbitrary position within the HTML form, write:

```
<input type="hidden"
name="require"
value="field1_field2_field3">
```

Note that in the standard settings “_” is considered as a nonvalue, so the same error page will be loaded if this value is encountered in a required item. To describe the precise missing item, another hidden item can be added (by default “requireErrMsg”), whose value is a list of error messages that are separated by the separator symbol in the order of the required items list—for example,

```
<input type="hidden"
name="requireErrMsg" value="You
did not reply to question 1_Reply
to 2?_Please answer question 3.">
```

The appropriate messages can be displayed in the error page by using the dynamic page manipulation for the field “requireErrMsg” (see Section 3.1.5).

Section 3.1.5

Dynamic Page Manipulation

Since *dynQuest* scans the HTML pages of the trial when sending them to the participant’s browser, it can manipu-

late the page contents. In its current version, *dynQuest* allows one to display the previous data of a study participant in the current trial page, which is a useful dynamic component for individualized interventions. This is implemented by using a new type of field—by default named “dynQuest.showData”—whose value is a field name. This field will be replaced with the item value in the data status (see Section 3.1.6). For example, if the study participant was previously asked for an activity with the item named “activity” and he or she answered “wall climbing”, then the HTML code (within the HTML form):

```
You decided to start <dynQuest
.showData="activity">. Where do
you want to do
<dynQuest.showData="activity">?
```

would be replaced by

```
You decided to start wall
climbing. Where do you want to do
wall climbing?
```

We remark that more sophisticated dynamic page manipulation can be realized—for instance, in combination with the free software PHP.

Section 3.1.6

Data Status

The data management of *dynQuest* is partly client based in the following sense. A hidden field is inserted into all pages that are sent to the participant’s browser containing the trial data submitted up to this point by the participant; we refer to this data as the *data status*. Upon submission of a form, the new data is appended to the data status, which is then written into the database (see Section 3.1.7). Note that the participant could theoretically manipulate the HTML pages and send arbitrary data to *dynQuest*—in particular, an arbitrary data status.

However, this mechanism avoids searching the database for previous entries to update a participant’s data status, and it makes the database a precise log of activity. This serves as a precaution against manipulation or loss of data in case the database was deleted before the participant ended the trial.

Section 3.1.7

The Database

All submitted data are stored in a text file, by default called “defaultDatafile.txt”. If this file does not exist, it is created automatically by using the list of field names (see Section 3.1.3). The location of the database is set in the definitions file; in particular, it can be in another, secure folder. We recommend making the database unreadable to Web clients.

The precise format of the database is as follows. The first line contains all field names separated by semicolons in the order of the list of field names. Five additional columns are inserted by *dynQuest*. The first column in the database is the participant’s code “dqcode” (see 3.1.2), and the last four columns are the name of the previous HTML file, the name of the current HTML file, the date and time

this entry was written into the database—for example, “15-Aug-2005 9:37:17”—and the version of *dynQuest* that wrote this entry—for example, “2005-Nov-9”.

Each line following the header contains the corresponding data of a participant as retrieved from her/his submissions. Note that in case of an existing database, *dynQuest* compares submitted items with the list of field names and, ignoring the structure of previous entries to the database, it simply appends a new line to the database. Since new lines are added each time a person submits a form, the database can be read as a log of activity, which allows one to find possible errors in the database. Typically, only one of the entries from a participant is relevant for later data analysis, and a natural choice is the data set with the largest number of nonempty items. This entry is usually (but not necessarily) the chronologically latest entry. The choice of entry with most data is implemented in the program “cleanQuest.pl.cgi” that can be run directly from a Web browser using its name as a URL—for example,

```
http://www.userpage.fu-
berlin.de/~slippke/dynQuest/
cleanQuestUnix.pl.cgi
```

More precisely, *cleanQuest* associates a data set to a participant if the code (see Section 3.1.2) and the first N items (e.g., sociodemographic data) coincide, where N can be set in the definitions file, but is zero by default. It then compares the number of entries of all associated data sets and keeps the one with the largest number of nonempty items. This algorithm is quite robust; for instance, participants can go back and forth in the trial with the browser without changing the outcome. A summary of the changes is displayed in the browser; see Figure 2.

Before purging the database, *cleanQuest* creates a backup copy of the untouched database with the same name plus the current date and time (see also Section 4.3). In addition, all changes to the database are recorded in a log file with the suffix “.log”. Hence, the default database file is purged, so that subsequent submissions are appended to a purged database. To avoid misuse, we recommend restricting access to the *cleanQuest* script. Note that it may be quite slow to clean large (several mega-

bytes) databases; in which case, we recommend moving the database to a different location so that *dynQuest* will automatically create a new empty database upon further submissions (see Section 4).

Section 3.1.8 Error Handling

If an internal error is encountered—for example, missing data in a link, a missing file, or an unreadable database—a brief error message describing the problem will be displayed. Additional text can be defined in the definitions file. By default, “Please contact” and an e-mail address set in the definitions file is displayed.

Section 3.1.9 Numerical Functions

The following four functions are currently implemented to perform special numerical computations whose results may be displayed on a follow-up page (see Section 3.1.5) or used for linking, also from the page on which the computation is done. The order of computation for several such fields on one page is BMI, summation, quotient, and range check.

Summation. The hidden form element with (default) name “dq-sum” allows one to add the values of any number of fields, whose names are listed (separated by the separator symbol) in the value of the form element after the field name in which the resulting sum will be stored. For instance, to write the sum of the fields “num1” and “num2” into the field “sum”, use

```
<input type="hidden" name="dq-sum"
value="sum_num1_num2">
```

Body mass index (BMI). The BMI = (weight in kg)/(height in cm)²/10,000 can be computed similarly by the hidden field with the (default) name “dq-bmi”. For instance, the form element

```
<input type="hidden" name="dq-bmi"
value="bmi_weight_height">
```

means that the BMI of the field values in “weight” and “height” will be stored in the field “bmi”.

```
This is cleanQuest Version 2005-Nov-12

Number of identifying entries: 0

Created a backup copy using the command:
cp defaultDataFile.txt "defaultDataFile.txt-2005-Nov-12_13:51:33"

Reading uncleaned database...

... read 8 entries. Writing cleaned database...

... done. Cleaned database has 2 entries.
```

Figure 2. Sample browser output of *cleanQuest*.

Range check. The hidden form element with (default) name “dq-range” allows one to check whether a value lies in a certain range/interval defined by (possibly fixed hidden) fields. For instance,

```
<input type="hidden"
name="dq-range"
value="range_num_low_high_1">
```

where “low” and “high” can be explicit numbers or names of fields, means the following. If the value in the field “num” is strictly larger than “low” and less than or equal to “high”, then “1” will be stored in the field “range” (else nothing happens). A sequence of such fields allows one to check several ranges. To check one inequality only, leave “low” or “high” empty; for example, “range_num_high_1” only checks whether “num” is less than “high”.

Quotient. Similar to the BMI computation, the quotient of two fields can be computed as follows (for zero denominator, the result is the text “NaN” for “not a number”):

```
<input type="hidden"
name="dq-quotient"
value="quotient_numerator_
denominator">
```

Section 3.2

Linking Pages

In this section, we discuss the possibilities to link pages of the trial. Subsections can be read in any order, and they are (1) unconditional links, (2) simple conditional links, (3) history-dependent links, (4) randomized links, (5) linking priorities, and (6) display the linking structure.

Section 3.2.1

Unconditional Links

The basic type of link is static so that the same follow-up page will always be loaded, independent of the answers. Such a link is programmed by inserting a hidden field element, by default with the name “next”, whose value is the name of the HTML file. In case this file name does not contain the HTML suffix set in the definitions file, the suffix will be added to the name. For example, to unconditionally link the current page to the page “node2.htm”, write (within the HTML form)

```
<input type="hidden" name="next"
value="node2.htm">
```

In case the suffix “.htm” is set in the definitions file one may equivalently use:

```
<input type="hidden" name="next"
value="node2">
```

Section 3.2.2

Simple Conditional Links

The simplest dynamic/conditional link depends on a single answer on the current page—that is, a single form element that can be activated by the participant. This is programmed by a form element whose name is the keyword for linking and the field name with the separator in between. For example, suppose a radio button should assign the value “1” to the field name “rglmt4” and link to the file “t4-10.htm”. The HTML code for this (in the default settings) is

```
<input type="radio"
name="next_rglmt4"
value="t4-10_1">
```

Instead of radio buttons, one can use text fields or pull-down menus. Table 4 lists the relevant properties of the form element for *dynQuest*.

Note that the last separator symbol in the value is assumed to separate the file name and field value. Hence, the value for the database cannot contain the separator symbol, but the file name can. For example, by default, “page_3_1” is interpreted as “page_3.htm” and field value “1”.

Pseudo-conditional link. A simple conditional link is, in fact, unconditional if it is contained in a hidden field, because these are always submitted. The difference to unconditional links is that for a pseudoconditional link, data for the given item is written into the database.

Detect missing values. Using the flow diagram, it is usually possible to reconstruct the items that a participant has seen, but left unanswered. However, this is not possible for randomized links. In general, it can be more convenient to obtain a missing value for unanswered items automatically, such as the value “99”. This can be realized by a pseudoconditional link, in addition to a visible simple conditional link. For instance, in addition to the previous example, use

```
<input type="hidden" name="next_
rglmt4" value="t4-10_99">
```

It is important that this hidden field lies after the visible field within the HTML form, because if the visible field is activated, then *dynQuest* will receive the values “t4-10.htm_1” and “t4-10.htm_99” for the same form element

Table 4
Properties of the Simple Conditional Link Field Element Relevant for *dynQuest*

Next	Linking Keyword for <i>dynQuest</i>
rglmt4	Name of the item (field name) in the field names file and database.
t4-10	Name without suffix of the next HTML page that will be loaded if this form element is activated and the page submitted.
1	Value of the item for the database, if this radio button was activated.

name “next_rglmt4”. By convention, the first of these will be used (see Section 3.1.1). By linking to an error page that describes the precise missing item, answers can be required more specifically than with the general require mechanism (see Section 3.1.4).

Section 3.2.3 History-Dependent Links

In this section, we describe a conditional link that requires possibly more than one participant’s answer (that might also go back to earlier pages) to match a prescribed list of values. For instance, if the participant’s gender was inquired on the first trial page, one can distinguish male and female participants on a later page of the trial without asking for the gender again.

The keyword for this link is, by default, “nextHistory”, and is implemented by a hidden field whose name consists of the keyword, a file name for the true case, and a file name for the false case separated by the separator symbol (by default, “_”). The hidden field’s value needs to be a list of pairs of field names and values that need to match for the true case. Here, field names and values need to have the separator symbol in between, and pairs are distinguished by two separators. For instance, the form element (within the HTML form)

```
<input type="hidden"
name="nextHistory_trueFile_
falseFile"
value="field1_value1__field2_
value2">
```

means the following. If the data for “field1” and “field2” in the data status (Section 3.1.6) precisely (!) match the values “value1” and “value2”, respectively, then the file “trueFile.htm” (can be any other name) will be loaded next; otherwise, it will be the file “falseFile.htm”.

To realize history-dependent branching for more than two alternatives, it is possible to omit either the entry “trueFile” or “falseFile”. To prescribe a link only for the true case, use the name

```
name="nextHistory_trueFile"
```

and for linking only in the false case, use (note there are two separator symbols)

```
name="nextHistory__falseFile".
```

Multiple linking can be implemented by a sequence of such single case links, which corresponds to a logical “or” combination. There must be a line break between each of these HTML form elements in the HTML file. To aid programming the trial, the processing of these links is displayed in the browser if a file named (by default) “showMsg.txt” exists.

Section 3.2.4 Randomized Links

Randomized control groups can be realized by a randomized choice of the next file from a list. This is implemented by a hidden field, by default with the name “randomize”, whose value is a list of file names with sepa-

rators in between. Each time the page is called, a file from this list is chosen pseudorandomly and is invisible to the participant by the PERL command “rand(k)”. Here, “k” is the length of the list, and the integer rand(k) is used as the index of the file in the order of the list. For example, if Participant A submits a form with

```
<input type="hidden"
name="randomize"
value="file1_file2_file3">
```

then the function rand(3) may yield the value 1, so “file1.htm” will be loaded. If Participant B submits the same form sometime later, rand(3) may be 3, so “file3.htm” is chosen, and so on. Note that uniform distribution of choices can only be expected for large numbers of participants.

Note also that a study participant might skip group-specific questions, so that the random group assignment would not be detectable. Therefore, we recommend introducing a variable that stores the group assignment via a hidden field (or a pseudoconditional link). For instance, if participants are randomized on page “t2_8.html” to either the page “t2_8a.html” or “t2_8b.html”, then the following construction gives a group assignment.

On page “t2_8a.html”: assign the value 1 to the variable “randomgroup” via

```
<input type="hidden"
name="randomgroup" value="1">
```

On page “t2_8b.html” assign the value 2 via

```
<input type="hidden"
name="randomgroup" value="2">
```

Section 3.2.5 Linking Priorities

If more than one linking command is given on a page, priorities decide upon the choice of link in the following order: (1) history-dependent link, (2) randomized link, and (3) depending on the naming and ordering in the HTML page, simple conditional links or unconditional links have priority 3, as discussed below.

As an illustration, if a history-dependent and a simple conditional link are both activated, then the history-dependent link will be chosen. However, care has to be taken within point 3 of the preceding list according to the conventions listed in Section 3.1.1. The choice of link depends on the ordering of the form elements on the HTML page and their names. Suppose the conditional link is activated. The resulting four cases are listed in Table 5.

Generally, if several simple conditional and unconditional links are submitted, then the last of these will be chosen; by HTML standard, the order in the HTML file

Table 5
Priority of Simple Conditional and Unconditional Links

Names/Order	Conditional First	Unconditional First
Equal	Conditional is chosen	Unconditional is chosen
Different	Unconditional is chosen	Conditional is chosen

is the order of submission. However, if several such links with the same name are submitted, then the first will be chosen. We remark that these priorities and ordering dependencies allow requiring items more specifically, and missing items can be traced (see Section 3.2.2).

Section 3.2.6 Display the Linking Structure

The program *showQuest* (in the scripts “showQuestPC.pl.cgi” and “showQuestUNIX.pl.cgi”) can be used to plot the complete linking structure of the trial—that is, the entire flow chart—by recursively analyzing the programmed links (see Figure 3). To use *showQuest*, set the file name of a trial page—usually the first—at the position described in the file “definitions.txt”. It is convenient to start *showQuest* directly from a Web browser so that the output is displayed in the browser window. For this, use the name as URL—for example, “localhost/dynQuest/showQuestPC.pl.cgi”. *showQuest* also creates a list of all field names that it encountered in the process and stores them in a text file named “fieldnames-sq.txt” by default. Note that *showQuest* can only find links that are programmed in HTML, as was described in Sections 3.2.1–3.2.3. This file can be used for the list of field names (see Section 3.1.3).

The structure of the trial can be recovered from this output, which thus allows checking for errors in link programming. For instance, the first line in the output in Figure 3 means that after page “index.htm”, the page “wave101.htm” always follows. (Between the first and second lines, some output is not shown here.) The third line means that a randomization occurs on page “wave112.htm”, and that the first group is sent to page “wave114a.htm”. The fourth line means that page “wave114a.htm” contains a simple conditional link, which links to “wave120.htm” if the value “1” is submitted for the variable “randomgroup”.

In fact, here, this link is a pseudoconditional link to store the group assignment (see Section 3.2.2). Several of the following lines refer to simple conditional links; for example, the last line means that page “wave114b2b.htm” links to “wave114b3.htm” under the condition that item “quiz2t1” was given the value “2”.

Note that *showQuest* makes assumptions about the form of the HTML file (see Section 3.1.1).

Often, one checks the trial by doing it, assisted by plotting the name of the currently used HTML file and the data that have been retrieved from the previous page. This is activated if files named (by default) “showPageNames.txt” and “showLastForm.txt”, with arbitrary contents, exist in the directory of the trial pages.

Section 4 Running a Trial in Practice

In the following, we will describe some aspects of practically running a PC- and WWW-based trial to help overcome initial barriers for beginning users.

Section 4.1 Checking the Layout

The appearance of the HTML pages needs to be programmed and checked by the user. This is supported by advanced HTML editors. Different browsers may display the same HTML page differently, which theoretically requires one to check the trial with different browsers and different default settings of these browsers. However, practically, these differences have played no great role in terms of usability of the trial for the study participants and interviewers.

Section 4.2 Preparation of the Trial

A typical barrier for the beginning user is how to install the software to make the trial work. As was mentioned in

```

This is showQuest Version 2005-Nov-14
line
1 index.htm:      :      ->   wave101.htm
...
2 wave111.htm:    :      ->   wave112.htm
3 wave112.htm(random) :      :      ->   wave114a
  wave114a.htm:    randomgroup:1 -> wave120
  wave120.htm:    : ->   wave121.htm
  wave121.htm:    :      :      ->   end
  wave112.htm(random) :      :      ->   wave114b
  wave114b.htm:    randomgroup:2 -> wave114b1
  wave114b1.htm:    quiz1t1:1   -> wave114b2b
  wave114b2b.htm:    quiz2t1:1   -> wave114b3b
  wave114b3b.htm:    quiz3t1:1   -> wave114b4b
  wave114b4b.htm:    :      :      ->   wave120
  wave114b3b.htm:    quiz3t1:2   -> wave114b4b
  wave114b3b.htm:    quiz3t1:3   -> wave114b4a
  wave114b4a.htm:    :      :      ->   wave120
  wave114b3b.htm:    quiz3t1:99  -> wave114b4c
  wave114b4c.htm:    :      :      ->   wave120
  wave114b2b.htm:    quiz2t1:2   -> wave114b3b
...

```

Figure 3. Sample (partial) output of *showQuest* from a trial flow diagram; at “...” part of the output is not shown in this figure; line information refers to the description in the text.

Section 2, there is no installation of *dynQuest* itself. All files simply need to be in a folder that can be accessed from a Web browser through the HTTP server and in which CGI scripts can be run. For the Apache server, this is usually the default setting for the “htdocs” folder and its subfolders; on a UNIX system, the administrator should be contacted.

In the Appendix, we describe how to check that the scripts work on your system. We emphasize that in order to use *dynQuest*, the trial pages must be accessed through the HTTP server—that is, the URL of the server—for example, “http://localhost/dynQuest/startpage.html” for a local server, and not “file:/...”.

A typical problem is that programming errors for the linking will result in wrong routes from one page to the next. To check the flow structure, one can simply click through the single pages to check them individually or use the program *showQuest* (see Section 3.2.6). For the latter, the start page must be defined in the definitions file. The program *showQuest* is started, for example, via “http://localhost/dynQuest/showQuestUnix.pl.cgi”.

Section 4.3 Data Management

Since *dynQuest* safely stores the data into the database, the remaining data management tasks for the user are to avoid loss and corruption of the database. This requires well-organized backups, which are partly done by *cleanQuest* when purging the database (see Section 3.1.7), and these backups can be further automatized by additional software. We recommend separating cleaned and uncleaned databases in different folders and media, and adopting a naming scheme using the date, time, and the person who stored or cleaned the database. Moreover, cleaning a large active database may cause a timeout on a browser that is awaiting data; thus, the size of the active database file should be kept small (as a guideline, less than a megabyte). This can be done either by regularly purging the database with *cleanQuest*, or moving by it to another location so that *dynQuest* will start a new database. Note that in the latter case, entries from the same participant may be spread over different files.

Section 4.4 Using the Data in Other Programs

Analyzing data requires appropriate software, and a barrier might be how to import the data into that software. Since the database is a text file in which items are separated by semicolons and data sets by line breaks (see Section 3.1.7), it can be easily imported into all common databases, spreadsheets, and statistics software, such as SPSS. However, doing so requires one to adhere to the rules of the software for importing data in this format. For instance, in SPSS, use the file menu “read text file” and follow the text import wizard. Default settings are appropriate, except for “Are variables included in the top of your file?” (choose YES), “Which delimiters appear between variables?” (no other than SEMICOLON), and “Specification for variable(s) selected in the data preview” (check whether data format is numeric, string, or date/time, and check number of characters). Note that syntaxes for data management can be saved for later use.

AUTHOR NOTE

We thank Ralf Schwarze for helpful comments, and Mary Wegner for her editorial assistance. J.R. acknowledges support by the NWO cluster NDnSt. Address correspondence to S. Lippke, Health Psychology, Freie Universität Berlin, Habelschwerdter Allee 45, 14195 Berlin, Germany (e-mail: s.lippke@fu-berlin.de).

REFERENCES

- BAILEY, R. D., FOOTE, W. E., & THROCKMORTON, B. (2000). Human sexual behavior: A comparison of college and Internet surveys. In M. H. Birnbaum (Ed.), *Psychological experiments on the internet* (pp. 141-168). San Diego: Academic Press.
- BARON, J., & SIEPMANN, M. (2000). Techniques for creating and using Web questionnaires in research and teaching. In M. H. Birnbaum (Ed.), *Psychological experiments on the internet* (pp. 235-265). San Diego: Academic Press.
- BIRNBAUM, M. H. (1999). Testing critical properties of decision making on the Internet. *Psychological Science*, **10**, 399-407.
- BUCHANAN, T., & SMITH, J. L. (1999). Using the Internet for psychological research: Personality testing on the World Wide Web. *British Journal of Psychology*, **90**, 125-144.
- CUMMINS, C. O., PROCHASKA, J. O., DRISKELL, M. M., EVERS, K. E., WRIGHT, J. A., PROCHASKA, J. M., & VELICER, W. F. (2003). Development of review criteria to evaluate health behavior change websites. *Journal of Health Psychology*, **8**, 55-62.
- EVERS, K. E., PROCHASKA, J. M., PROCHASKA, J. O., DRISKELL, M. M., CUMMINS, C. O., & VELICER, W. F. (2003). Strengths and weaknesses of health behavior change programs on the Internet. *Journal of Health Psychology*, **8**, 63-70.
- GOSLING, S. D., VAZIRE, S., SRIVASTAVA, S., & JOHN, O. P. (2004). Should we trust web-based studies? *American Psychologist*, **59**, 93-104.
- KIRSCH, S. D. K., & LEWIS, F. M. (2004). Using the World Wide Web in health-related intervention research. *Computers, Informatics, Nursing*, **22**, 8-18.
- KRANTZ, J. H., BALLARD, J., & SCHER, J. (1997). Comparing the results of laboratory and World Wide Web samples on the determinants of female attractiveness. *Behavior Research Methods, Instruments, & Computers*, **29**, 264-269.
- LIPPKE, S., SNIEHOTTA, F. F., & LUSZCZYNSKA, A. (2005). Social cognitions across the stages of behavior change. A comparison of two stage models. *Polish Psychological Bulletin*, **36**, 43-50.
- LIPPKE, S., ZIEGELMANN, J. P., & SCHWARZER, R. (2004). Initiation and maintenance of physical exercise. *Research in Sports Medicine*, **12**, 221-240.
- MAR, C. M., CHABAL, C., ANDERSON, R. A., & VORE, A. E. (2003). An interactive computer tutorial to teach pain assessment. *Journal of Health Psychology*, **8**, 161-173.
- MORROW, R. H., & MCKEE, A. J. (1998). CGI scripts: A strategy for between-subjects experimental group assignment on the World-Wide Web. *Behavior Research Methods, Instruments, & Computers*, **30**, 306-308.
- NIGG, C. R. (2003). Technology's influence on physical activity and exercise science and application: The present and the future. *Psychology of Sport & Exercise*, **4**, 57-65.
- SCHWARZER, R., SCHÜZ, B., ZIEGELMANN, J. P., LIPPKE, S., LUSZCZYNSKA, A., & SCHOLZ, U. (2007). Adoption and maintenance of four health behaviors: Theory-guided longitudinal studies on dental flossing, seat belt use, dietary behavior, and physical activity. *Annals of Behavioral Medicine*, **33**, 156-166.
- WESTEN, D., NOVOTNY, C. M., & THOMPSON BRENNER, H. (2004). The empirical status of empirically supported psychotherapies: Assumptions, findings, and reporting in controlled clinical trials. *Psychological Bulletin*, **130**, 631-663.
- WILLIAMS, J. E., MCGRAW, K. O., & TEW, M. D. (1999). Undergraduate labs and computers: The case for PsychExps. *Behavior Research Methods, Instruments, & Computers*, **31**, 287-291.
- WOLFE, C. R. (1992). Using Authorware Professional for developing courseware. *Behavior Research Methods, Instruments, & Computers*, **24**, 273-276.
- WOLFE, C. R., & REYNA, V. F. (2002). Using NetCloak to develop server-side Web-based experiments without writing CGI programs. *Behavior Research Methods, Instruments, & Computers*, **34**, 204-207.

APPENDIX**Checking the Scripts on Your Computer**

1. Run “perl -d <scriptname>” from the prompt where “<scriptname>” is any dynQuest file with the suffix “.cgi”. You may have to use a path, e.g., “c:\apache\perl\bin\perl -d <scriptname>”. Check for error messages.
 2. Check the location of the PERL executable on your system. For the PC version, the setting is “c:\apache\perl\bin\perl.exe”, for UNIX “/usr/bin/perl”. If these settings are not valid on your system, open all *dynQuest* scripts in a text editor and adjust these in the first line of each “.cgi” file.
 3. For the UNIX version, also check that the commands “/usr/local/bin/lockfile -6 -r 10” and “/sbin/rm -f” work. These manage attempts of simultaneous access to the database. If the paths or commands on your system are different, change the settings in the file “definitions.txt” accordingly.
-

(Manuscript received December 20, 2005;
revision accepted for publication April 1, 2006.)