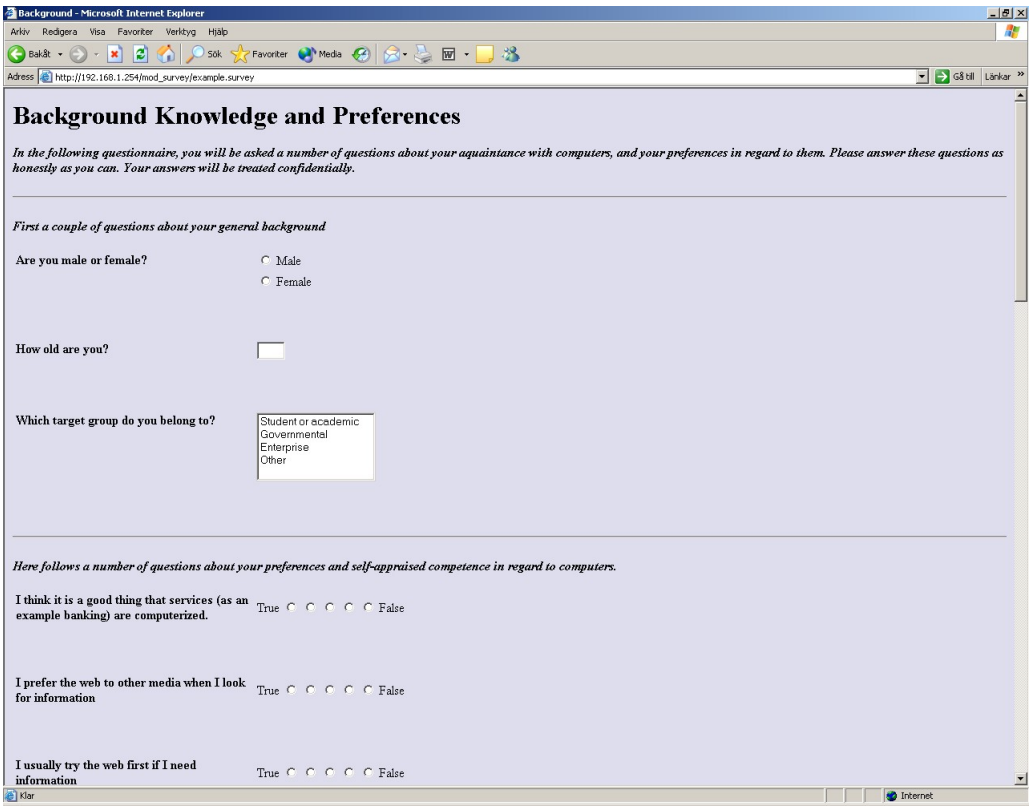
Mod_Survey for Beginners

3.2.x branch



MSc. Joel Palmius
Mid Sweden University
Department of Information Technology and Media

Mod_Survey for Beginners: 3.2.x branch
by MSc. Joel Palmius

DocBook Version Edition
Published 2004

Table of Contents

List of Figures

Chapter 1. Introduction

Welcome to the comprehensive user's guide to using Mod_Survey. This chapter outlines what web questionnaires in general and Mod_Survey in specific is all about, and gives a short introduction to the contents and layout of the manual.

1.1. What is a Web Questionnaire?

A web questionnaire is a questionnaire the respondent can answer via the web, using his normal web browser.

1.1.1. The Incentive for Web Questionnaires

Questionnaires is a good way to measure things quantitatively, at least when they work out ok. But everyone who has done a bigger survey, say a questionnaire with a hundred questions sent to two hundred persons, knows that the next step in the process might be tedious to say the least.

What you, as a questionnaire researcher, will realise when the respondent answers start to arrive, is that the digitalization of the paper form requires a lot of resources. In the normal low-budget case, you will have to manually type in the answers into your statistics program. This effectively limits the maximum size of many surveys: The resources needed both for sending out paper questionnaires and then typing them in becomes overwhelming.

Now, this can be solved with expensive machinery: There are optical readers which can automatically copy answers from papers into the computer. However, the cheaper alternative is to use web questionnaires. With a web questionnaire, the costs are no longer related to the size of the sample. It does not cost more to collect 3000 answers than 2000, since the answers are automatically stored in the web questionnaires database, immediately ready for analysis.

1.1.2. When and How Can Web Questionnaires be Used?

As more and more potential respondents both have access to and competence enough for using the Internet, the use of web questionnaires becomes more attractive. In principle, web questionnaires can be used as a replacement for other forms of surveying as soon as it can be ensured that the respondents are able to reach and use the web questionnaire via the Internet.

Web questionnaire surveying can be conducted both on known samples and unknown samples. With a known sample, it is available digitally to the respondents as soon as they have been made aware of the questionnaire's URL. This can be distributed to the respondents in many forms: It

could be sent via e-mail or even printed on paper. With an unknown sample it is possible to make a questionnaire publically available on the web and let anyone who sees it answer.

1.1.3. When Should Web Questionnaires not be Used?

The great draw-back with web questionnaires is of course its dependency on technology. It does not lend it self easily to surveying in low-tech groups such as elderly people. Also, the internet penetration does not necessarily imply good internet competence: There is a technological barrier to accessing and answering an online survey even for some people who have an internet connection.

Further, while the online technology makes interesting survey features available, it may also make some things less stable. The web questionnaire software might contain bugs, the responstimes might be unbearably long, the respondent's screen resolution is unknown and the respondent might have a faulty or old web browser. Web surveying introduces sources of error which are not there when doing paper-based surveying.

1.1.4. Scientific Aspects of Web Questionnaires

As a researcher you should also be aware that there are statistically significant issues with using web questionnaires. It is not necessarily so that web questionnaires give less true answers as compared with paper questionnaires, but they are noticably different.

These issues are called mixed mode problems. The experimental setup for measuring these differences are as follows: Select a computer-aware population, for example students. Pick two equal samples from this population. Give one sample a questionnaire printed on paper. Give the other sample the questionnaire online. The questionnaire is exactly the same on paper and web, the paper version is optimally a printed version of the web questionnaire. Chances are you will notice a statistically significant difference in answers.

It is beyond the scope of this manual to describe these phenomena in detail, but as a hint, you would probably notice that the web questionnaire sample tend to avoid extremes (in a scale from true to false, they will keep closer to the middle than in the paper sample).

Tip
There is a lot of interesting litterature concerning the mixed mode issues. Although most concern other combinations than web/paper, a number of master and bachelor theses have been written specifically concerning web and paper. Some of these are available on Mod_Survey's home page.

Apart from the above, most litterature on questionnaires in general should also apply to web questionnaires. Also paper questionnaires need be formulated to fit the target population. The additional steps of pilot testing the questionnaire would need a web usability test (testing how the questionnaire looks on different platforms and different hardware).

1.2. What is Mod_Survey?

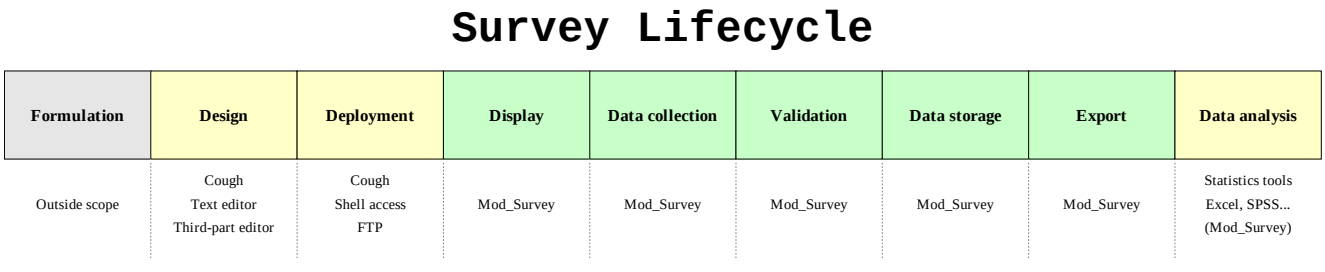
Mod_Survey is a software which enables the use of web questionnaires. Optimally it is installed on one of your organization's web servers. With Mod_Survey available, you can conduct any amount of web questionnaire surveys and have data and responses handled automatically server-side.

The explicit purpose of Mod_Survey is to enable advanced questionnaire technology, even in the cases where this will lead to a more difficult learning curve. Mod_Survey will be described in more detail in the next chapter.

1.2.1. Where does Mod_Survey Fit In?

Mod_Survey and other software (such as Cough) related to Mod_Survey handles most of the life cycle of a web questionnaire, but not everything. Below is an image describing this life cycle. The grey parts are entirely outside the scope of Mod_Survey. The yellow parts are partially handled by Mod_Survey or related software. The green parts are entirely handled by Mod_Survey.

Figure 1-1. Survey Lifecycle



Please note that for space reasons several important aspects of surveying have been excluded from this figure. Most notable is the respondent management (such as population, samples, contacting respondents, reminders and so forth), which would be gray or yellow if placed in the figure.

Formulation: The formulation of the survey is currently entirely outside the scope of Mod_Survey. The formulation is the scientific base of the questionnaire, things like dependent and independent variables, indexes, question texts, question order and so forth. Mod_Survey expects you to know how you want the survey to be formulated before beginning. As a side note, ideas for a formulation support has been discussed, but there has so far been no attempt at actual implementation. A support of this kind could look like a standard software wizard ("Which are your independent variables?", "How many dependent variables do independent variable 1 influence?"...).

Design: Design is partially inside the scope of Mod_Survey. The difference between formulation and design is that the design is the actual visual appearance in the end. The Cough add-on layer includes a simple survey editor which can help you with the actual survey layout. Without Cough, surveys are formulated as XML file via any standard text editor.

Deployment: Deployment is the actual publication of the survey, making it available to the respondents via the net. In practise this means putting a survey file somewhere on a web server. This is supported through an upload form in Cough. Without Cough, upload is done via standard unix tools (such as FTP, SCP...). Without Cough, deployment will require shell access on the Mod_Survey server.

Display: When deployed, the questionnaire is automatically formatted and displayed in the respondents web browser, when the respondent requests access to the survey. Technically this is a transformation from XML to HTML, but the author of the survey does not need to be aware of this process as Mod_Survey takes care of all the details.

Data collection: When the respondent has filled out the survey and clicks "submit", Mod_Survey handles the data input. The survey author does not need to be aware of the details here either, since everything is handled transparently.

Validation: If requested by the survey author, the collected data can be format validated. Examples of supported validaton are that a question must be answered, that a submitted value can only be within a certain span, and that crap characters cannot be submitted. Per default all checking is disabled. If an error is found, the respondent is notified and is asked to correct the input and re-submit.

Data storage: When collected and validated, the data is stored in a central repository. The survey author does not need to care about this, since it is handled automatically. However, if so wished, this behavior can be overridden, so that data is saved in a RDBMS or in a text file.

Export: When all answers are collected, the survey author can download the data in a number of popular formats. Mod_Survey automatically transforms and formats the data to these formats when requested.

Data analysis: Mod_Survey has very minor support for data analysis. In practise it is expected that the survey author wants to download the data and open it in a third-part software such as Excel or SPSS. The previous step (export) supports formatting data into files possible to open in these packages.

1.3. About this manual

Before beginning with the manual as such, let’s state some things about its format.

1.3.1. Admonitions

Throughout the manual, four kinds of admonitions will be used: Tip, Caution, Importand and Note.

Tip
A "Tip" is a hint about something which might be interesting for you, but which isn’t entirely necessary to solve the core task at hand.

Caution
A "Caution" is information about something that might possibly lead to damage to the data. This might be where you might cause data corruption or when something will hinder respondents from answering the survey.

Important
An "Important" is information about something you must be aware of to be able to solve the core task. Failing to follow an Important text might cause various problems for you, but not to the degree that it will damage your data.

Note
A "Note" is information of something that you need be aware of to do an efficient job, but which can be ignored if you know what you are doing.

1.3.2. Terminology

In the following, a "survey" shall denote one web questionnaire file written using the Mod_Survey syntax. A questionnaire with several pages is a "survey chain". The "user" is the author of the survey, usually the analyst. The "respondent" is the person answering the survey. The "administrator" is the person responsible for the server where Mod_Survey is installed.

1.3.3. Not Included in the Manual

The manual’s purpose is to provide an *introduction* to Mod_Survey. Most features will be covered briefly here. However, the manual does not cover the full syntax. More advanced users of Mod_Survey will likely want to read *Mod_Survey Syntax Reference*, which is a terse no-nonsense reference to the full syntax.

Further, the manual is directed towards *users* of Mod_Survey. Thus system administrator tasks such as backups, upgrades and security will not be covered here. These issues are covered in *Mod_Survey Sysadmin Guide*.

Finally, there is nothing about code structure and about the behind-the-scenes of Mod_Survey. This information, which might be important for programmers and people who want to extend Mod_Survey, is available in *Mod_Survey Programmer’s Guide*.

Chapter 2. About Mod_Survey

This chapter provides a brief introduction to Mod_Survey, in order to make the reader aware of what to expect from it, and perhaps more important what to not expect.

2.1. A Brief History

The history of Mod_Survey started in the autumn 1998. I was studying my third year at the information systems science program at Mid Sweden University, and was out on the second trainee period. This time I worked at a local newspaper, and my task was to examine their intranet solution and suggest improvements for it.

The first problem arising was the data collection. Having earlier conducted a paper questionnaire at a big company, I knew it was a nuisance having people fill out a form and send it back. Somehow those papers always got lost or were placed in the bottom of a heap on a desk. Besides, at the newspaper project we were short on time and resources. I decided that since everyone had access to their local intranet server, a CGI script would make things considerably easier and more efficient.

As I had been hobby-programming web applications for some time, throwing a custom CGI script together did not take a long time, and it was successfully put to use for the data collection part of the project. It was ugly and crude, but it worked. I found it so convenient that I decided to make the script a bit more general so I could use it again in later surveys. This was the birth of Mod_Survey and can be said to be version 0.0, although I hadn’t named the application then.

A year later I had added to and formalized many parts of the script. It could now handle several surveys and it knew how to save files both as semi-colon delimited fields and in a database. I named the script "Survey", and it was used in a couple of projects at the university. For example it was used in a usability study as a tool for collecting user feedback. As the users in this case were people with various handicaps and low education, it was shown that web questionnaires were not limited to perfectly healthy people with a degree in computer science. The code in the script was still rather un-organized and the questionnaire description format was crude: the surveys were defined as tab-delimited fields in text files. However, it worked and looked reasonable nice. This was version 1.0.

After having finished my degree in spring 2000, I continued to work at the university, partly with various project and partly as a post-graduate student. I and the guy who had done the handicapped-people thesis continued to tinker with the script. It was restructured from the bottom and this time implemented in mod_perl. The name was changed from "Survey" to "Mod_Survey". Things got very much faster and more convenient, but eventually we ran into a dead end, mainly because of the decision to keep the old tab-delimited fields definition format. This was version 2.0.

While the rewrite was not an unqualified success, it had the good consequence that we were noticed by a large governmental organization, NIWL, the National Institute for Working Life. As

they conducted really large paper-based surveys and then hired several persons to type the results into SPSS, they saw the potential of web surveys. In the end they sponsored me and my colleague to develop the system into a usable application which would be flexible enough to use in their surveying. I had actually begun thinking about another rewrite, but with the funding we got, the rewrite could get ambitious.

During the autumn 2000, the code was re-organized again and put in a much better structure. The definition file format was changed from tab-delimited fields to XML and most basic question types were implemented. A user of current-date Mod_Survey would recognize most of the way it looked. In december 2000 Mod_Survey 3.0.0, as it was now officially called, was deployed for NIWL's survey. The system delivered up to expectation, and a home-page was created for downloading code and documentation.

As a part of the contract with NIWL, it has been stated that the final Mod_Survey would be licensed as GPL and copyrighted by me. With the first home-page release on december 12 2000, Mod_Survey could be said to be an open-source project. However, the project was not announced publically until april 2001, which was the date when version 3.0.4 appeared on Freshmeat.

The project plodded on with occasional updates. By summer 2002 new features such as multipaging, multiple-answer CHOICE and the MEMO field had been added. The application was used here and there and users started to subscribe to the mailing list and occasionally submit code patches and ideas. Local students started to do projects and theses around web questionnaires in general. Most notably Cecilia Bäckström and Christina Nilsson did an ambitious study of Mixed Mode problems.

During the summer, Mathieu Jan of the Sympa crew joined in the Mod_Survey development and started to send in ambitious updates. He contributed important updates such as the internationalization layer, and the ability for survey persistence. All in all it took a while to consolidate the changes, but by early spring 2003, the 3.0.x branch reached its current state with 3.0.14. Since then only minor additions have been made to that branch.

During spring 2003 a gang of italians from Demetra and YaaCs took up the flag and started to send in ambitious ideas and code changes. Since Mod_Survey had grown in a evolutionary manner and had gotten a bit cluttered, and since the new ideas required significant changes, Mod_Survey was branched into what was to become the 3.2.x branch. In the late summer 2003 a first alpha release of 3.2.0 was made available. This contained advanced features such as conditional branching, scripting and dynamic contents.

An ambitious EU project was planned for developing Mod_Survey further, with participants from universities and companies in italy and sweden. In the end the application was never sent in, partly because of economy and partly because of health problems. However, many ideas had arisen during the discussions and the italian gang, mainly through BugAnt, continued as heavy contributors to the development.

Development snapshots of 3.2.0 continued to appear during spring 2004 and many behind-the-scenes infrastructural code aspects were rewritten to be more logical and consistent. By

mid summer a first public beta was released. Mid Sweden University held a summer course in "Web Questionnaire Design", which used a beta of 3.2.0 as base technology.

At the time of writing, 3.2.0 final is close to release, and chances are that you are using either a late beta or the final version.

2.2. Different Versions of Mod_Survey

At the time of writing, there are two major versions of Mod_Survey: 3.0.x and 3.2.x. The 3.0.x branch is the code that have lived more or less in the same state since autumn 2000. It is what most users of Mod_Survey still use for production use. 3.2.x is the next stable branch of Mod_Survey. It contains major feature, security and stability upgrades. This is what users are encouraged to use if they are trying to choose between 3.0.x and 3.2.x.

3.2.x is not backwards-compatible with 3.0.x and following manual is completely based on the assumption that you are using 3.2.x. The instructions are not relevant for 3.0.x.

Aside from the major versions, the CVS repository also contains a 3.1.x branch which has been discontinued. There is no reason whatsoever to use this. It is only mentioned here so you know that you should avoid it.

2.3. Mod_Survey’s Goals and Target Group

Before reading the list of features and details of Mod_Survey, you should probably be aware of what the goals of Mod_Survey are, and who the intended users are.

From the beginning, the main objective of Mod_Survey has been feature completeness. The ideal is a survey suite which supports all the advanced questionnaire components and designs the user could conceivably desire. If an ability is not natively supported, then at least Mod_Survey’s code base should be clear and logical enough to be extended in an efficient manner.

Ease of learning ("learnability" to use Nielsen’s usability terminology) is a secondary goal. Feature completeness and richness of advanced features often conflicts with learnability. Complexity of output will often require complexity of input. Mod_Survey’s main mode of input is currently through editing files with a certain XML-based tag notation. While being harder to learn than a traditional GUI, I still feel that the expressive power of this markup would be hard to capture through a GUI. For the same reason I also write all HTML code by hand rather than using a WYSIWYG editor such as Dreamweaver or Frontpage.

The focus on expressive power and feature-completeness is interdependent on Mod_Survey’s target group. The target group is users who need to conduct larger surveys, who needs to have

powerful interactions with statistics suites such as SPSS, and who will need to get their hands dirty in customizing output and format of their web questionnaires.

Mod_Survey can certainly be used for simpler questionnaires, such as a one-question poll or vote questionnaire. It might be considered a bit of overkill, but there’s nothing stopping you from using it as such.

2.4. What does Mod_Survey Support?

Features and extensions has been added to Mod_Survey for years, and there are a lot of small seemingly minor features that most users will never need. These features are documented in the syntax reference. The following is a list of some of the "larger" features, mainly there to provide some kind of overview of what to expect from the system.

2.4.1. Basics

Question types: Most basic question types are supported. This includes multiple-answer choice blocks, multi-line open answers and matrixes.

Data exporting: Once data has been collected, the user can choose to export it into a multitude of data formats, suitable for import in most major statistics suites. Export formats include but are not limited to delimited fields, fixed columns, data scripts and tables. Import support has been tested and found working with suites such as MS Excel, MS Access, MiniTab, SPSS and R.

Answer checking/constraints: Questions can optionally have checking, to make sure that the respondent has answered a question and/or inputted a correctly formatted answer.

Descriptives: The export features includes viewing descriptive statistics and frequency tables calculated from the submitted data.

Randomization: Question blocks of the matrix type can be randomized internally both over rows and columns (or both at the same time).

2.4.2. More advanced data collection

Date and time: Date and time features include both timestamping an answer, and measuring the time it took for a respondent to answer a survey.

Server interconnection: Environment variables set by the Apache server can be added to the data. This makes it possible to, for example, record the IP or the browser application of the respondent.

Completely custom questions: If the available question types are not sufficient, the user can design completely custom questions which can use the whole spectrum of HTML and JavaScript.

2.4.3. Database interconnectivity

Optional DBI backend: Answers can optionally be recorded in any DBI-compatible database manager, such as PostgreSQL, Oracle or MySQL.

Importing database fields: Data can be fetched from any DBI-compatible database manager, and can appear both in respondent-visible output, or as submitted together with the respondent's answers.

2.4.4. Modularization

Conditional branching and routing: The respondent can be routed to different questions based on previous answers. The routing supports advanced aggregate boolean expressions.

Linking and fragments: Fragments (such as a page header) can be broken out from the survey design and placed in separate files for run-time inclusion into the surveys.

Enabling and extending exports: The data exports modules can be individually enabled or disabled or added to. Each possible data export is written as a separate file building on a powerful API.

2.4.5. Layout customization

Themes: There are several layout- and color themes following the base installation.

Stylesheets: Most aspects of the layout can be customized through specifying external stylesheets.

Graphics and extra HTML: Extra html tags for boldfacing words or inserting images can be added in the respondent-visible output.

2.4.6. Other advanced features

Dynamic contents: Respondent-visible output can be automatically modified depending on earlier answers, for example in order to include the respondent's name or prefix titling with "Mr" or "Mrs".

Perl scripting: It is possible to include perl "snippets" in the survey files, to extend their functions with hand-written perl scripts.

Mailing data copies: It is possible to automatically mail copies of all respondent answers to several email addresses.

Detailed access restriction: Access to various aspects of a survey (such as answering, data export or administration) can be set both on a user and a host basis.

2.4.7. System meta

Apache-based: Mod_Survey is an extension module to Apache, the world's most popular web server software.

Free: Mod_Survey is open-source and is thus free both in regards to content and cost.

2.5. What does Mod_Survey *not* support?

There are also a number of features of surveying which is either in queue for addition, or outside the scope of Mod_Survey.

No GUI for editing surveys: The first thing new users will discover is that there is no graphical user interface for editing survey files. Users are expected to write surveys in Mod_Survey's tag markup using any text editor of choice. (A GUI is/will be a part of Cough, an add-on layer to Mod_Survey, but will probably never be a part of Mod_Survey as such)

Persistance: Data persistence has been supported in earlier versions of Mod_Survey but has not yet been ported to the latest version. Data persistence is the respondent option of saving data temporarily in order to return later to finish the survey.

Second-tier data modification: There is no functionality for modifying submitted data. Mod_Survey is a data collection tool, and modifying already submitted data is currently outside the scope. However, in the long term this is a planned feature.

Random order between questions: Randomization is only supported inside matrix blocks. Apart from that the order of questions cannot be randomized (this is planned for the immediate future).

Population management: There is no functionality for selecting samples or keeping track of a population. This is outside the scope of Mod_Survey as such, but is planned for inclusion in Cough.

2.6. Comparing Mod_Survey with Other Systems

Mod_Survey is not unique in providing the ability to deploy web questionnaires. There are many systems doing this. These systems can be categorizes in four categories: Embedded poll systems, survey hosters, forms APIs and web questionnaire suites.

Embedded poll systems:

Survey hosters:

Forms APIs:

Web questionnaire suites:

2.7. Getting Access to Mod_Survey

This manual presupposed that Mod_Survey is already installed on a server you have access to. You need login information to that server, and you can likely get these from your local system administrator. As soon as you have access to such a server, and are able to place files in the web tree somehow, then you are ready to go.

Note that while the above in most cases means that you need shell access (the ability to log in via a console prompt) this is not entirely necessary. Many deploy survey files by saving them on a network drive via the common windows network. Your system administrator will know which of these approaches is the most convenient for you.

Chapter 3. Basic Work with Survey Files

The following instructions should enable you to perform the basic tasks needed for writing and using a survey.

Note
The following instructions do not take the Cough add-on layer into account. Many of these tasks are much easier when using Cough, which is summarily described in a later chapter in this manual. However, even if you are going to use Cough, the following sections are useful to read and understand. Thus it is recommended that you skim them through before starting with Cough.

3.1. Requirements

Before you begin writing survey files, you will need a server running apache and mod_survey. Further you need some kind of access to it. If you are able to put a normal homepage on the server, you should have access good enough to put survey files on it too.

To write the survey files you can use any text editor, ranging from notepad in windows to vi in unix. Many text editors have syntax highlighting for XML, which might come in handy when writing survey files.

3.2. Writing a Survey File

Once you have made sure you fulfill the basic requirements, you are ready to start writing your first survey file. To begin with, start your favorite text editor, or if lack of one, start notepad in windows.

A basic survey file consists of tags, making it look a bit like a HTML page. Every survey starts and ends with SURVEY tags. A first survey might look like this:

```
<SURVEY TITLE="My first survey">
  <TEXT NAME="name" CAPTION="What is your name?" />
</SURVEY>
```

In the example we have created a survey with the title "My first survey". It has one question, a open answer text field with the question text "What is your name?". When furter treated, we will refer to this question using the id "name".

You will here not that tags come in two different forms. You have both the *open* tags which are written in the form <TAG>..</TAG>, and the *closed* tags which are written in the form <TAG .. />. The difference between this is that the open tags may contain other tags, while the closed can not.

Once you have made sure that the survey file seems properly written, you should save it to disk. Save it with the name "first.survey" (and make sure you do not save it as "first.survey.txt" or "first.txt", it is important that it ends with ".survey").

With the file saved on disk you have essentially done what is required to define a very basic survey. Next is to deploy it online.

3.3. Uploading a Survey File

To deploy your survey file, you need to place it somewhere in the web tree. This means it should be placed on the server in a directory that the Apache web server software can find. Usually, this is is a sub-directory of your home directory. The sub-directory's name varies, but is usually called "www" or "public_html".

3.3.1. Using the windows network

If you are on a windows machine and have access to a network drive mapped to your home directory on the server, things become very easy. Imagine that your home directory is mapped as H:. Now simply move the file you saved ("first.survey") to the www subdirectory on the network drive, usually H:\www\.

Even if the network drive is not mapped, you might be able to save files to the web server via the windows network. Imagine that your server is called "webserver.domain.com", and that your username is "joepal". Now open a file window (double-click on "my computer"), and write "\\webserver.domain.com\joepal" in the address field. With some luck you will see the contents of your home directory pop up, and there find a sub-directory called "www" or "public_html". Simply move the file your saved to this sub-directory.

3.3.2. Using SSH/SCP

(to be written)

3.3.3. Using FTP

(to be written)

3.3.4. Writing surveys on-site

(to be written)

3.4. Accessing a Survey

Survey files are accessed in the same way as common web pages. The server will take care of the formatting, and present the respondent with a web questionnaire.

Supposing you have successfully placed your survey file in the web tree, you can now direct your browser to it, to see how it looks online. Start your browser (Internet Explorer, Mozilla or what browser you use), and edit the address field. Suppose your server address is "webserver.domain.com", that your user name is "joepal", and that your survey file is still called "first.survey". Then accessing the survey could be done through writing the following in the address field of your browser:

```
http://webserver.domain.com/~joepal/first.survey
```

Now, several things might happen when you access the survey. The best is if you see a web page containing one text input field and the question "What is your name?", along with two buttons ("Submit" and "Clear"). This would mean that everything works and that you have successfully deployed your first survey. However, one of a number of errors might also occur:

Page says Error 404, not found. This means both that Mod_Survey is not installed, and that your survey file was not found. See below.

Source of survey is displayed rather than a questionnaire. This means that Mod_Survey is not installed on the web server. You will need to speak with your system administrator to solve this.

Page says Document error, not found. Mod_Survey works, but your survey file could not be found. Chances are that you either misspelled the name or put the survey file in the wrong place.

Other Document error. Mod_Survey works and your survey was found, but it contained an error, probably some small spelling mistake. The error message should be detailed enough to tell you what the mistake was.

3.5. Administrating a Survey

Once deployed, the survey should more or less take care of itself. However, each survey has an administrative page where you can, for example, remove all submitted data. Note that most administrative functions are switched off per default, for security reasons. For now, we will only see how to access the administrative page, but later on you may wish to read the chapter "Restricting Access" to see how to enable the administrative functions.

All functions of a survey are reached by adding arguments to the survey. The most common functions are "admin", which we will demonstrate here, and "data" which will be demonstrated in the next section. Actually, simply viewing a survey is the function "display", although if not specified, the system will assume that this is what you wanted.

To see the administrative page of a survey, you need to specify that the desired action is "admin":

```
http://webserver.domain.com/~joepal/first.survey?action=admin
```

Note that we are still using the same address, but have added "?action=admin" to it. With some luck, you will be presented a menu with several options when having entered this.

The menu options should be quite self-explanatory, so we will be satisfied with the above for now.

3.6. Downloading Data

Once some respondents have answered the survey, you will likely want to download the submitted data to, for example, view it in a statistics program.

Reaching the data function works in the same way as accessing the administrative function. You will simply have to specify that the desired action is "data":

```
http://webserver.domain.com/~joepal/first.survey?action=data
```

Again, you will be presented with a menu. This time you will get the option of downloading the data in any of several possible formats. The data downloading is described in more detail in the chapter "Downloading and Using Survey Data".

Chapter 4. Writing Questions

The most basic use of Mod_Survey is to present a questionnaire online. Most basic questionnaire object types are supported. This chapter summarizes the basic question objects.

4.1. What is a Question?

A "question" in Mod_Survey's sense is a subset of the possible "variables" or "objects", namely the objects that have a lead-in question text, and a field for the respondent to answer the question.

Thus, a *question* is what would be called a question on a paper questionnaire. Each question is represented as a *variable*, or in other words a field with the ability to get different values depending on what the respondent answers. Note that there can also be variables that do not have questions: For example, recording the IP address of the respondent is a variable, but not a question, since the respondent is never formally asked for it. An *object* is a tag in the survey source. An object can contain several variables and/or questions (as in the case of MATRIX), or contain a single variable/question, or contain no variable at all. Objects can be purely passive or only affect layout.

A *question object* is an *object* which contains one or more *questions*, and one or more *variables*. This chapter only deals with question objects.

4.2. Closed Single-Choice Questions

There are three basic question objects for defining closed single-choice questions: The BOOLEAN, the CHOICE and the LIST.

The BOOLEAN is a question with a true/false answer, represented as a checkbox.

The CHOICE is a list of possible answers presented as a column of radio buttons and labels. All possible answers are always visible.

The LIST is a list of possible answers presented either in a listbox or in a drop-down menu. If the number of possible answers is larger than the height of the listbox, the respondent can scroll down to find more answers.

4.2.1. BOOLEAN

The BOOLEAN is one of the simplest question types available in the repertoire. It consists of a lead-in question text and a checkbox, which can either be checked or unchecked. Thus the BOOLEAN always represent a true/false condition.

One example of a question for which the BOOLEAN would be fit is the standard "Yes, I want to be notified via e-mail":

```
<BOOLEAN NAME="mail" CAPTION="Yes, I want to be notified via e-mail" />
```

It is also possible to set the initial state of the checkbox via the CHECKED parameter, which can either be "yes" or "no" (defaulting to "no").

4.2.2. CHOICE

The CHOICE is one of the most versatile components available. In its simple state is represents choice between a limited set of options. The CHOICE also supports multiple-select and open answers, but these features will be treated further down.

The CHOICE is different to the BOOLEAN in the way that it is an "open" tag, meaning it contains subtags. Each possible option is represented as a CHOICEELEMENT tag. A very easy example would be the close-to-obligatory question about whether the respondent is male or female:

```
<CHOICE NAME="gender" CAPTION="Are you male or female?">
  <CHOICEELEMENT VALUE="0" CAPTION="male" />
  <CHOICEELEMENT VALUE="1" CAPTION="female" />
</CHOICE>
```

CHOICE do also, as most but not all objects, support the MUSTANSWER parameter. Setting the MUSTANSWER parameter to "yes" will force the respondent to choose one of the options. If MUSTANSWER is "no" (default) then the result will be given the value of the parameter ILLEGALVAL (defaulting to "-1").

4.2.3. LIST

While the CHOICE is intended for a limited set of options, resulting in an integer representation, the LIST is usable for giving the respondent a larger list of options, represented as strings. The LIST ends up as either a "listbox" or as a "combobox", thus only displaying a limited set of options at a time but giving the user the ability to scroll down to find more options.

For example, a LIST could be used in a question about which department an employee belongs to on a company. For space reasons, the following only includes three departments, but there would not be a problem to add a hundred more.

```
<LIST NAME="dept" CAPTION="Where do you work?">
  <LISTELEMENT CAPTION="ITM" />
  <LISTELEMENT CAPTION="SHV" />
  <LISTELEMENT CAPTION="SOA" />
</LIST>
```

The number of options presented at a time is decided by the VISIBLELEN parameter, defaulting to "5". If set to "1", the LIST will end up as a "combobox".

In the above the LISTELEMENT tags get the value of their CAPTIONS, but it is also possible to set a VALUE parameter explicitly in the same way as in the CHOICEELEMENT. This might be a good idea if the options in the LIST are represented as long strings.

Tip
This might be a good place to re-iterate that this manual only deals with the basics of Mod_Survey syntax and markup. The question objects mentioned here contain many more features than the ones listed above. For example it is possible to randomize options, state if a variable should be alphanumerical or numerical, and so on. After having understood the basics, it might be a good idea to thumb through the "Mod_Survey syntax reference".

4.3. Multiple-Choice Questions

There are two ways to represent multiple-answer questions in Mod_Survey: Via the MULTI parameter in CHOICE, and via the MULTI parameter in MATRIX. The MATRIX tag is treated further down.

In its most basic form, the CHOICE only allows one answer, but if the MULTI is set to "yes", it will allow multiple answers. One example could be the following:

```
<CHOICE NAME="pizza" CAPTION="Add what ingredients on my pizza:" MULTI="yes">
  <CHOICEELEMENT VALUE="1" CAPTION="Olives" />
  <CHOICEELEMENT VALUE="2" CAPTION="Curry" />
  <CHOICEELEMENT VALUE="3" CAPTION="Bearnaise" />
  <CHOICEELEMENT VALUE="4" CAPTION="Banana" />
</CHOICE>
```

The respondent will be presented with a number of checkboxes and can check the ones he thinks applies. Note that the MUSTANSWER/ILLEGALVAL behavior (see above) still applies.

4.4. Open Questions

The manual has so far only dealt with closed question, questions where the survey author has pre-determined all possible answers. However, it is often useful to let the user add completely free answers. A question which allows textual input is called an "open" question.

4.4.1. One-line text fields

The most basic open question is the one-line text field, which can be used to, for example, ask the respondent about his name:

```
<TEXT NAME="name" CAPTION="What is your name?" />
```

This will display a text input field, into which the respondent can enter up to 80 characters. The maximum length of the field can be varied with the MAXLEN parameter (which defaults to "80").

4.4.2. Numerical fields

One special case of open answers are format-checked text input fields which are used for querying for numerical answers. These are also handled by the TEXT object. For example, it can be used to ask the respondent for his age:

```
<TEXT NAME="age" CAPTION="How old are you?" NUMERICAL="yes" />
```

In the above, the NUMERICAL parameter has been set to "yes", telling the system to perform format checking on the answer. It is also possible to set, for example, maximum and minimum allowed answers.

4.4.3. Multiple-line text fields

Sometimes a one-line text field is too limited. Of course, there is nothing stopping the survey author from setting the MAXLEN parameter to quite a large number. However, a small input field

psychologically implies that the answer should be short.

To fetch more verbose answers from the respondent, it is thus useful to allow a larger text field, which allows line-feeds:

```
<MEMO NAME="comment" CAPTION="Please add your thoughts about the above" />
```

The MEMO object does not have any size limitation, which may lead to very long answers being accepted. This might be cumbersome in data analysis, but most data exports allow for filtering out MEMO fields when downloading data.

4.4.4. Otherfields

It is often useful to combine closed questions with an open option. In Mod_Survey, the CHOICE object supports being combined with an open answer. This can be used when none of the options applies, or when the respondents wants to add another option.

As an example we an extend the pizza question from a previous section:

```
<CHOICE NAME="pizza" CAPTION="Add what ingredients on my pizza:"
MULTI="yes" OTHERFIELD="Other, please specify:" >
  <CHOICEELEMENT VALUE="1" CAPTION="Olives" />
  <CHOICEELEMENT VALUE="2" CAPTION="Curry" />
  <CHOICEELEMENT VALUE="3" CAPTION="Bearnaise" />
  <CHOICEELEMENT VALUE="4" CAPTION="Banana" />
</CHOICE>
```

In the above, the respondent would be presented with a list of checkboxes, and a text field.

4.5. Scales

Scales can be represented in two ways in two ways: Discreet and continuous. This is an important decision for the survey designer, as it has a serious impact on applicable methods for data analysis.

The discreet scales are handled by CHOICE, LIST and MATRIX. A discreet scale is a scale with answers which can be compared and put in ascending or descending order, but which are not continuous. A non-discreet scale is handled by LICKERT, and is a continuous variable which is applicable for data analysis such as means and distributions.

Note
The spelling of the LICKERT object is indeed LICKERT with "ck". The most common spelling in literature is "likert scale", without the "c". When starting the work of Mod_Survey I had a statistics book which either used a very odd spelling or misspelled the word. Now, several years later the "ck" spelling is so ingrained in the system that it would be a pain to change it.

4.5.1. Discreet scales

A discreet scale is simply a set of options arranged according to value. A question regarding frequency of an event might be a discreet scale, if available answers are given in the form of "less than once a week" and "once a year". Since the scale is not continuous, data analyses such as means and distributions are not applicable.

Syntactically, there is no difference between a CHOICE or LIST defining a discreet scale, and a CHOICE or LIST defining any other question. To continue the event question example, it could look like this as a CHOICE:

```
<CHOICE NAME="often" CAPTION="How often do you read mail?">
  <CHOICEELEMENT VALUE="0" CAPTION="Very, very often" />
  <CHOICEELEMENT VALUE="1" CAPTION="Once every ten minutes" />
  <CHOICEELEMENT VALUE="2" CAPTION="Once an hour" />
  <CHOICEELEMENT VALUE="3" CAPTION="Once a day" />
  <CHOICEELEMENT VALUE="4" CAPTION="Once a week" />
  <CHOICEELEMENT VALUE="5" CAPTION="Very, very seldom" />
</CHOICE>
```

Or as a list:

```
<LIST NAME="often" CAPTION="How often do you read mail?">
  <LISTELEMENT VALUE="0" CAPTION="Very, very often" />
  <LISTELEMENT VALUE="1" CAPTION="Once every ten minutes" />
  <LISTELEMENT VALUE="2" CAPTION="Once an hour" />
  <LISTELEMENT VALUE="3" CAPTION="Once a day" />
  <LISTELEMENT VALUE="4" CAPTION="Once a week" />
  <LISTELEMENT VALUE="5" CAPTION="Very, very seldom" />
</LIST>
```

Matrices also very often define discreet scales, but they will be treated in the section about matrices further down.

4.5.2. Continuous scales

A continuous scale is a position between two extremes, for example an answer between "Agree totally" and "disagree completely". A continuous scale is usually implemented as a LICKERT in Mod_Survey. A basic example could be:

```
<LICKERT NAME="know" CAPTION="I know a lot about computers" />
```

...which will result in a five-step scale between True and False. The values to the left and to the right of the scale are regulated with LEFTTAG and RIGHTTAG respectively, and the number of steps with STEPS.

Note also, that with some care and designing you could formulate a CHOICE or LIST question as a continuous scale. It is mainly an effort of providing possible answers which would fit as scale positions.

4.6. Matrices

A MATRIX is different from the question objects we have seen so far in the way that it is a *set* of questions sharing format and layout, rather than a single question. Matrices are often used in questionnaires when the possible answers to a set of questions are always the same. Using a matrix relieves the respondent of having to read through the possible alternatives for each questions, which may make the respondent feel that the questionnaire is easier to overview and fill out.

4.6.1. The basic MATRIX

The basic MATRIX is a set of rows (MATRIXROW) defining questions, and columns (MATRIXCOLUMN) defining possible answers. The MATRIX has a NAME parameter, but the variables will be the rows automatically named as the NAME parameter with a number added. One easy example would be:

```
<MATRIX NAME="color" CAPTION="Which color do you want...">
  <MATRIXROW CAPTION="..the border to be?" />
  <MATRIXROW CAPTION="..the background to be?" />
  <MATRIXROW CAPTION="..the foreground to be?" />
  <MATRIXROW CAPTION="..the text to be?" />
  <MATRIXCOLUMN CAPTION="Red" VALUE="1" />
  <MATRIXCOLUMN CAPTION="Green" VALUE="2" />
  <MATRIXCOLUMN CAPTION="Blue" VALUE="3" />
  <MATRIXCOLUMN CAPTION="Yellow" VALUE="4" />
```

```
</MATRIX>
```

This would leave us with a four by four matrix with the variables color01 .. color04, and the possible answer values 1 .. 4.

Important
Mod_Survey currently imposes a length restriction on variable names. No variable name can be longer than 8 characters. Since MATRIX automatically adds two characters for numbers, this means that the NAME parameter can only contain six characters.

4.6.2. MATRIXes as scales

Defining a discreet scale in a MATRIX is of course mainly a matter of formulating relevant answer alternatives. Consider, for example, the following:

```
<MATRIX NAME="fpsjuk" CAPTION="During the last 12 months I have had...">
  <MATRIXCOLUMN VALUE="4" CAPTION="Often" />
  <MATRIXCOLUMN VALUE="3" CAPTION="Quite often" />
  <MATRIXCOLUMN VALUE="2" CAPTION="Sometimes" />
  <MATRIXCOLUMN VALUE="1" CAPTION="Seldom" />
  <MATRIXCOLUMN VALUE="0" CAPTION="Never" />
  <MATRIXROW CAPTION="trouble with neck or back" />
  <MATRIXROW CAPTION="trouble with muscles or limbs" />
  <MATRIXROW CAPTION="allergical trouble" />
  <MATRIXROW CAPTION="dry skin or dry mucous membranes" />
  <MATRIXROW CAPTION="cold or other respiratory infections" />
</MATRIX>
```

Continuous scales might of course be a bit more problematic, but there is nothing stopping you from using 1,2,3..N as CAPTIONs in the columns.

4.6.3. Further reading

The MATRIX block supports many advanced features, such as randomizing rows and/or columns, defining variables among the columns, multiple-answers restricted against either or both axes and more. Please refer to the syntax reference for more information about this.

4.7. Designing Custom Question Types

If you are not satisfied with the available set of question objects, Mod_Survey also supports defining completely custom question objects. In these you simply state which variables they define, and then draw the question with HTML of your own. One example would be asking for a direction:

```
<CUSTOM VARIABLES="cu1" ESCAPED="no">
  <b>I want to go...</b><br /><br />
  <table width="300" height="300" cols="3" rows="3" border="1">
    <tr>
      <td> </td>
      <td>
        <center>
          <input type="radio" name="cu1" value="n" /><br />North
        </center>
      </td>
      <td> </td>
    </tr>
    <tr>
      <td>
        <input type="radio" name="cu1" value="e" /> West
      </td>
      <td> </td>
      <td>
        East <input type="radio" name="cu1" value="e" />
      </td>
    </tr>
    <tr>
      <td> </td>
      <td>
        <center>
          South<br /><input type="radio" name="cu1" value="s" />
        </center>
      </td>
      <td> </td>
    </tr>
  </table>
</CUSTOM>
```

Important
Using the ESCAPED="no" setting makes writing the survey files by hand a lot easier, but it also breaks XML compatibility. If you ever plan to open your survey file in an XML editor, you should use ESCAPED="yes" and escape all HTML markup (ie < should be written <).

If your CUSTOM block defines a set of variables, the VARIABLES parameter should include a comma-separated list of them.

Chapter 5. Colors, Fonts and Decoration

(to be written)

5.1. Static Lead-in Texts

It is very often useful to have a static lead-in text in a questionnaire, such as a title or a few lines with instructions. Writing such static text is done through using CUSTOM but without setting its VARIABLES parameter. As seen in the previous section about CUSTOM, any HTML markup can be used. For example:

```
<CUSTOM ESCAPED="no" >
  <h1>My survey</h1>
  Please don’t lie to me when answering this survey.
  <br />
  <br />
  <hr />
  <br />
</CUSTOM>
```

Advanced users may also use the CUSTOM blocks for influencing the style of following standard question objects, but that is an altogether higher school of HTML.

5.2. Themes

Mod_Survey will do most of the layouting for you, and if you don’t specifically ask for something else, the layout will be made to look like the rest of Mod_Survey (ie the Data menu and the Admin menu). However, the default layout theme "cloud" is just one of many others.

If you want to try looking at other themes, you can change the THEME parameter of the SURVEY tag. Available themes are as of writing this text:

cloud (default)	Theme which boxes questions for redability.
formal	Plain gray, black and white theme made for looking formal.
slate	Clean black and white, no decoration whatsoever
invert	Same as slate but white on black.
rose	Pink and red theme with larger fonts.

The theme can also be overridden when displaying the survey file. If you normally call the survey at:

```
http://127.0.0.1/mysurvey.survey
```

You can explicitly say you want an other theme than the one specified in the survey file:

```
http://127.0.0.1/mysurvey.survey?action=display&themeoverride=rose
```

If you are uncertain about how things look in the respondents' web browsers, it might be a good thing to provide links with different theme overrides.

5.3. Stylesheets

The theme engine of Mod_Survey is entirely based on style sheets. In practise what happens when you specify the THEME parameter, is that one of the corresponding css files will be linked as a stylesheet. These stylesheet are all available in the Mod_Survey web root, in the system sub directory. For example, you could read the Cloud theme file (cloud.css) via:

```
http://127.0.0.1/mod_survey/system/cloud.css
```

..assuming your server is at 127.0.0.1 and that your browser do not attempt to do anything strange when displaying CSS files.

5.3.1. Extending the theme

The SURVEY tag sports a STYLESHEET parameter through which you can specify a stylesheet of your own. The order of the inclusion is that the theme stylesheet will be linked to first, and the STYLESHEET stylesheet second.

5.3.2. Replacing the theme

While adding new styles will be sufficient for most people, it can also be useful to completely override the THEME and *only* use styles of your own. To disable the inclusion of the THEME stylesheet, set THEME to "external". Then specify a STYLESHEET of your own. The easiest way to make a new stylesheet is probably to copy the one you think look best of the available theme sheets and then modify it.

The formulation of good theme stylesheets for Mod_Survey is by no means any exact science. All visible components are divided into DIV tags, and the advanced user will probably want to look at

the HTML source output to figure out which styles to define or change.

5.4. Font Styles

You can easily specify font styles for any part of the visible output, for example if you want to boldface a word in a sentence. For example

```
<TEXT NAME="subj" CAPTION="What is your {b}main{/b} subject?" />
```

The supported styles are bold ({b}..{/b}), underline ({u}..{/u}) and italics ({i}..{/i}).

5.5. Colors

In the same way that you can specify font styles, you can also change the color of parts of the output. For example:

```
<TEXT NAME="col" CAPTION="This sentence contains {red}red text{/red}." />
```

All colors follow the same pattern, ie {color}..{/color}. The defined colors are red, green, blue, yellow, purple, cyan, black and white.

5.6. Graphics

Almost any visible output, such as a CAPTION, an element in a CHOICE or a static lead-in text can also contain graphics. These are in practise linked-in images, which in HTML would be specified with the <IMG> tag. In Mod_Survey you use the {image} markup:

```
<TEXT NAME="smile" CAPTION="This is a smiley: {image}happy.png{/image}" />
```

5.7. Extra HTML

Finally, it is also possible to instert completely custom HTML markup in any visible output, but it must be escaped. This makes writing it a bit cumbersome, but altogether possible. For example, let us assume that you want to insert a break-row (
) tag in a caption:

```
<TEXT NAME="q1" CAPTION="Question 1: {[}br /{]}How are you?" />
```

In other words, a < is written as {[} and a > as {]}. Other notable escape codes are {’} for ", and {A} for &. A full reference of codes are available in the syntax reference.

Chapter 6. Collecting Automatic Data

(to be written)

6.1. Non-Visible "Questions"

So far, we have mainly dealt with question objects that are visible to the respondent, or in other words "questions". However, not all variables need to be set by the respondent explicitly. In several instances, it can be useful to collect information from the respondent automatically, or even without his knowledge.

Some basic examples of such automatic data is to store a timestamp of when the questionnaire was submitted, to record from which IP the answers came, or perhaps even which browser the respondent used.

6.2. Constant Values

The simplest form of automatic data is the CONSTANT object. This is used to store a constant field, which is not possible for the respondent to influence. This could for example contain the latest revision of the questionnaire file:

```
<CONSTANT NAME="revision" VALUE="2004-09-13 / JP" />
```

With this set, each record would contain a string field called "revision" which would have the value of the CONSTANT's VALUE parameter (as of the time when the respondent submitted his answers). Changing the VALUE parameter only influences new records, old records keep the value as it was when they were submitted.

6.3. Fetching Data from the Environment

One of the more interesting features of the automatic data collection is the reading of environment variables through the ENV tag. The ENV tag in itself is rather simple, it only specifies a name and a field. However, apache sets a multitude of environment variables automatically, and these are all available to the ENV tag.

Keeping a reference of all environment variables set by apache is beyond the scope of this manual (see apache's documentation instead), but the following examples might be enlightening:

```
<ENV NAME="ip" FIELD="REMOTE_ADDR" />
<ENV NAME="browser" FIELD="HTTP_USER_AGENT" />
<ENV NAME="server" FIELD="SERVER_NAME" />
```

The above would create three string variables. "ip" would contain the IP address of the respondent. "browser" would contain information about the respondent's web browser. "server" would contain the name of the server running the Mod_Survey software (perhaps not all that useful).

6.4. Date, Time and Timing

There are two tags related to time and timing in Mod_Survey: TIMER and DATETIME. The first measures the time it takes for the respondent to fill out a survey page, while the second inserts a timestamp.

6.4.1. Measuring response time

To measure the time it took to fill out a survey page, use the following:

```
<TIMER NAME="p1time" />
```

The resulting value is an integer value representing the time in seconds it took between the load of the page and the submit of data.

6.4.2. Inserting date and time

The DATETIME tag is the all-round date- and timestamping tool. It supports inserting date and time information in a number of formats. In its most basic use, it inserts date and time in common central-european format:

```
<DATETIME NAME="now" />
```

This would result in a string looking like "2004-12-04 13:48:50". This is probably what most people would want to use. However, this format might make it difficult to, for example, measure the time between two different respondents. Consider then the following:

```
<DATETIME NAME="now" FORMAT="#e" MAXLEN="20" />
```

This will result in "now" getting the value of "seconds since epoch", which is 1970-01-01 00:00. The equivalent of the previous date example would be roughly "1102251173".

Tip

The syntax reference contains a list of valid format codes. There are currently about 12 of these available.

Chapter 7. Surveys with more than One Page

One of the more interesting features of web surveying, is the possibility to route respondents to only the questions relevant to them. In a paper questionnaire, you would typically formulate question in the form "if yes on the above question, how often did this happen". With a web questionnair you have the option to simply not show this question if the respondent answered "no".

In Mod_Survey routing is handled between "survey pages". All you have seen so far is contained within one survey page. The following expands this to allow you to display several pages in an optionally non-linear manner.

7.1. Basics of Multi-Paging

All multi-page surveys begin with one page, written in the same manner as you have seen so far. If nothing is added to it, it is considered a single-page survey, so in order to enable multipaging we have to add a *routing tag*. The routing tag determines which page should be the next to display once the current page has been answered.

There are currently four types of routing tags: unbranching ("ROUTE"), simple conditional ("CASEROUTE"), complex conditional ("IFROUTE") and random route ("RANDOMROUTE"). The first three will be treated below, while the random route will be treated in the chapter about randomization.

Finally, to tie things together, the last page in the chain must have a SEQUENCE tag, containing a list of which pages the respondent could possibly have seen.

In multipaging, no data is saved in the intermediary pages: All data is pushed to the last page. Thus, to access data, you should add ?action=data *to the URL of the last page in the chain*

Note
To summarize: 1, All pages but the last must contain a routing tag. 2, The last page must contain a SEQUENCE tag. 3, Data is considered to be saved in the last page.

7.2. Unbranching Multi-Paging

The most basic form of multipaging is the straight route, with no branches. In this, all pages are

displayed in order and no pages are skipped. Imagine that this is page1.survey:

```
<SURVEY TITLE="First page">
  <TEXT NAME="name" CAPTION="What is your name?" />
  <ROUTE CONTINUE="page2.survey" />
</SURVEY>
```

And that this is page2.survey:

```
<SURVEY TITLE="Second page">
  <TEXT NAME="email" CAPTION="What is your email address?" />
  <SEQUENCE SELFINCLUDE="yes">
    <FILE FILENAME="page1.survey" />
    <FILE FILENAME="page2.survey" />
  </SEQUENCE>
</SURVEY>
```

With this we have set up a very simple multi-page sequence. The first page contains the question "what is your name?", and once answered, the second page is displayed. This page contains the question "What is your email address?".

Technically, the first page specifies how to continue with the ROUTE tag, which always contains the parameter CONTINUE. The second page lists all possible parts of the page sequence in the SEQUENCE tag. The SELFINCLUDE says that we intend to include the page containing the SEQUENCE tag in the list for readability.

7.3. How Variables are Handled

Before continuing with the more complex types of routing, we will want to know how variable data might end up. Basically, on a single-page survey a variable can end up in two ways: answered or not answered. Implicitly "not answered" could be expanded into "seen but not answered".

In branching multipaging, there is a third possibility: that a variable was not answered simply because the respondent was never routed to the page containing the variable. Thus the variable can also have the state "unanswered because not seen".

The practical values for these states will vary depending on your survey settings. The defaults for the two not-answered options is "-1" for "seen but not answered". This is changeable through setting ILLEGALVAL in each relevant question. Note that is MUSTANSWER="yes" in a question, it will never get ILLEGALVAL. The default for "unanswered because not seen" is 999, and is set globally through the NOTDISPLAYEDVAL parameter of the SURVEY tag.

Not all data exports attach any significance to these values, but for example the SPSS export will explicitly export "system-missing" for these values so that they are excluded from calculations.

7.4. Conditional Branching with CASE

If you have a simple decisive variable, such as a true/false, yes/no or male/female relation, then the CASEROUTE can be applied easily. The CASEROUTE routing tag is relevant when there are a finite number of discreet options, such as the result of CHOICE or LIST. Imagine that you want to ask different questions to men and women:

```
<CHOICE CAPTION="Are you male of female?" NAME="sex">
  <CHOICEELEMENT CAPTION="male" VALUE="0" />
  <CHOICEELEMENT CAPTION="female" VALUE="1" />
</CHOICE>
<CASEROUTE SWITCH="sex" DEFAULT="error.html">
  <CASE VALUE="0" CONTINUE="male.survey" />
  <CASE VALUE="1" CONTINUE="female.survey" />
</CASEROUTE>
```

With this, the next page would be "male.survey" if the respondent answered male, and "female.survey" is the respondent answered female. If none of the options were chosen, the next page would be "error.html".

It is recommended, but not necessary, that all routed end up in the same final page (see also the section on how to fail with multipaging). This last page should contain a SEQUENCE tag as seen in the previous section. This tag should list *all possible* pages.

7.5. Conditional Branching with IF

If you want to do more complex routing, such as on the combination of two different variables, then the IFROUTE is for you. The IFROUTE is able to specify a number of boolean conditions which should be fulfilled for a routing to happen. For example, imagine that you want to ask different questions to men, women, boys and girls, in essence route on both age and sex:

```
<TEXT NAME="age" CAPTION="age" NUMERICAL="yes" MUSTANSWER="yes" />
<CHOICE CAPTION="gender" NAME="gender" MUSTANSWER="yes">
  <CHOICEELEMENT CAPTION="male" VALUE="0" />
  <CHOICEELEMENT CAPTION="female" VALUE="1" />
</CHOICE>
```

```
<IFROUTE DEFAULT="error.html">
  <IF THEN="male0-15.survey">
    <LESSTHAN VARIABLE="age" VALUE="16" />
    <EQUALS VARIABLE="gender" VALUE="0" />
  </IF>
  <IF THEN="female0-15.survey">
    <LESSTHAN VARIABLE="age" VALUE="16" />
    <EQUALS VARIABLE="gender" VALUE="1" />
  </IF>
  <IF THEN="male16-25.survey">
    <LESSTHAN VARIABLE="age" VALUE="26" />
    <MORETHAN VARIABLE="age" VALUE="15" />
    <EQUALS VARIABLE="gender" VALUE="0" />
  </IF>
  <IF THEN="female16-25.survey">
    <LESSTHAN VARIABLE="age" VALUE="26" />
    <MORETHAN VARIABLE="age" VALUE="15" />
    <EQUALS VARIABLE="gender" VALUE="1" />
  </IF>
</IFROUTE>
```

Here we can see that we can end up in five different places depending on the answers of two variables (if none of the four boolean combinations is fulfilled, we end up in "error.html"), There are a number of possible boolean conditions. These can be found in the syntax reference.

7.6. How to Fail Miserably with Multipaging

The above has demonstrated some of the capabilities available in regards to multipaging. It might be tempting to make very complex routing chains where a number of question blocks are hidden or shown depending on answers. Technically, this is no problem, as long as you are able to specify what you want to happen.

However, you should consider if there is a really good purpose with doing a branching multipaging? Does it really make the questionnaire easier to answer for the respondent?

Further, if you have several very similar questions, you might want to consider reformulating them so that they end up being the same variable. This might make data analysis a lot easier.

Finally, you will likely want to think carefully before you make routes that end up in different final destinations. *You will get one data respository per possible final destination.* Thus, things will be a lot easier for you if you manage to route all respondents to the same final page.

From my own experience, I know it can be frustrating to have collected a lot of data in several repositories just to find out that they really should be treated as one material and thus be merged in

the final statistics program, something which of course then was a lot of unnecessary work. Having made a sane routing, I would have gotten all data in one place from the start.

Chapter 8. Dynamic Contents

The dynamic contents consists of a number of things, which here has been arbitrarily grouped into a chapter. The following is only a brief introduction, since particularly the perl snippet functionality can be used to make very advanced extensions to the surveying functionality. Describing this would be beyond the scope of this manual.

8.1. Referencing Previous Answers

One thing which has a certain usability effect on respondents, is the functionality of personalizing the questions. Imagine that a respondent already has answered that he is male, and that his name is Joel. In consequent questions, we could now very well address him with his name and with "mr.". Imagine that page1.survey looks like this:

```
<SURVEY TITLE="dynamics part 1" >
  <TEXT NAME="name" CAPTION="My name is" />
  <CHOICE NAME="gender" CAPTION="I am a">
    <CHOICEELEMENT VALUE="0" CAPTION="man" />
    <CHOICEELEMENT VALUE="1" CAPTION="woman" />
  </CHOICE>
  <ROUTE CONTINUE="page2.survey" />
</SURVEY>
```

Then we can write a question in page2.survey, looking like this:

```
<CHOICE NAME="grown" CAPTION="{!gender/0:Mr,1:Mrs!} {$name$}, do you consider you
  <CHOICEELEMENT VALUE="0" CAPTION="yes" />
  <CHOICEELEMENT VALUE="1" CAPTION="no" />
</CHOICE>
```

Here we can see three different ways of referencing previous variables. The first {!..!} is a *selection*, which outputs a different value depending on the value of a variable (in this case "gender"). The second {$..$} references the *value* of a variable, in this case "name". The thirst {%..%} references the *caption or label* corresponding to a value of a variable.

8.2. Including External Files

Sometimes it is convenient to include external fragments into your survey file. This can be done *pre-parse* or *display-time*

8.2.1. Pre-parse include

Including a file pre-parse means that is is included before the survey file is interpreted. The survey system will then treat the linked fragment as if it was a part of the survey file. Imaging that your company has a lot of surveys which include the same question. Rather than writing this question in each and every survey, it is linked to, so that a change in the central place can propagate to all the surveys.

For example, this common question could be on which department you are working, in a file called "/central/survey/dept.question":

```
  <LIST NAME="dept" CAPTION="Where do you work?">
    <LISTELEMENT CAPTION="sales" />
    <LISTELEMENT CAPTION="marketing" />
    <LISTELEMENT CAPTION="development" />
    <LISTELEMENT CAPTION="maintainance" />
    <LISTELEMENT CAPTION="support" />
    <LISTELEMENT CAPTION="human-resource" />
  </LIST>
```

Then the fragment could be linked into a survey file looking like this:

```
  <SURVEY TITLE="A company question">
    <TEXT NAME="name" CAPTION="What is your name?" />
    {@/central/survey/dept.question@}
  </SURVEY>
```

File names can be absolute or relative

8.2.2. Display-time include

In some instances, you do not want to include markup that should be interpreted. You might simply want to break out long chunks of text to be placed outside the survey to make it easier to edit. For example, imagine that you want to ask a respondent if he agrees to a license file (which is here placed in "/central/survey/license.txt"):

```
  <SURVEY TITLE="EULA">
    <CUSTOM ESCAPED="no">
  <pre>
{¤/central/survey/license.txt¤}
  </pre>
    </CUSTOM>
```

```
  <CHOICE NAME="eula" CAPTION="Do you agree to the above?">
    <CHOICEELEMENT VALUE="0" CAPTION="No" />
    <CHOICEELEMENT VALUE="1" CAPTION="Yes" />
  </CHOICE>
</SURVEY>
```

This would simply paste the contents of the linked file into the visible output. This is generally faster than a pre-parse include, although neither should incur any significant overhead.

8.3. Perl Snippets

Another feature of dynamic contents is the internal perl scripting. In essence, this is a perl script embedded in the survey file, which can be run before parse, at display time or at submit time. This can be used for implementing more advanced behavior in the survey. However, an overview of perl scripting is outside the scope of this manual.

The syntax reference contains information about the necessary markup, but you will probably not want to experiment with this unless you know what you are doing.

Chapter 9. Restricting Access

(to be written)

9.1. Setting up User Files

(to be written)

9.2. Access Levels

(to be written)

9.3. Managing Respondents and Unique Answers

(to be written)

Chapter 10. Using Databases

(to be written)

10.1. Why Would You Want to Use Databases?

(to be written)

10.2. Setting Up DBI

(to be written)

10.3. DBI Tweaks and Syntax

(to be written)

10.4. Letting Respondends Delay their Answers

(to be written)

Chapter 11. Downloading and Using Survey Data

(to be written)

11.1. The Data Module

(to be written)

11.2. The HTML Export

(to be written)

11.3. The SPSS Export

(to be written)

11.4. Importing Data into Excel

(to be written)

11.5. Importing Data into Access

(to be written)

11.6. Importing Data into SPSS

(to be written)

11.7. Importing Data into MiniTab

(to be written)

11.8. Importing Data into R

(to be written)

11.9. Importing Data into a RDBMS

(to be written)

Chapter 12. Troubleshooting

(to be written)

12.1. I Only See the Source

(to be written)

12.2. Internal Server Error

(to be written)

12.3. Document Error: Not Found

(to be written)

12.4. Permission Denied

(to be written)

12.5. Getting More Help

(to be written)

Chapter 13. Acknowledgements

(to be written)

13.1. About the Author

(to be written)

13.2. People Who Contributed to Mod_Survey

(to be written)

Appendix A. Short Installation Instructions

(to be written)

Appendix B. Migration Guide for 3.0.x Users

(to be written)

Appendix C. The GNU General Public License

(to be written)

Appendix D. Contact, Feedback and More Information

(to be written)