



Technical guide to producing online questionnaires: Print version

This module was written by Rob Shaw except where stated in section headings or case studies.

For information on copyright and how to cite this module, please refer to the following web pages:

- Copyright statement: <http://www.geog.le.ac.uk/orm/site/copyright.htm>
- Citation policy: <http://www.geog.le.ac.uk/orm/site/citation.htm>

Contents

1. Aims and learning outcomes	2
2. Introduction to online questionnaire production: Overview and options	3
3. Choosing software for online questionnaire production	14
4. Using software for online questionnaire production	22
5. Introduction to HTML 1	31
6. Introduction to HTML 2	41
7. Introduction to CSS	56
8. Web forms	69
9. Introduction to JavaScript	79
10. Form validation	89
11. Key design issues	134
12. Gathering information about participants	145
13. Server-side processing	173
14. Frequently-asked questions	186
15. Glossary	192
16. Resources	200

Aims and learning outcomes

Aims

1. To provide background information on the range of options available for the production of online questionnaires and the key benefits and drawbacks of each;
2. To provide an introduction to the range of software and services available to assist in the production and implementation of online questionnaires;
3. To provide an introduction to HTML and CSS;
4. To introduce web forms and explore the technology behind the effective design of online questionnaires;
5. To introduce and allow practice in the use of JavaScript for basic form validation, and the gathering and preservation of user information;
6. To provide basic guidelines for the use of server-side technologies for the collection and processing of data;
7. To discuss relevant issues when choosing off-the-shelf software for the creation and administration of online questionnaires, and to provide general guidelines on how such software is used;
8. To describe key terms, definitions and terminology in relation to the technical aspects of online questionnaires;
9. To provide links to additional resources, frequently asked questions and print versions.

Learning outcomes

At the end of this module, you will be able to:

1. Identify appropriate technical options for the implementation of online questionnaires according to the context of research;
 2. Identify key issues to bear in mind when choosing off-the-shelf software or services for online questionnaires;
 3. Identify the technical features required for your online questionnaire, and select suitable software and/or services to provide for these requirements;
 4. Use a range of technologies in the creation of online questionnaires, such as HTML, CSS and JavaScript;
 5. Create a web form, test it for consistency, and incorporate effective navigation and a progress bar;
 6. Add basic validation and a means of collecting and preserving information about the user;
 7. Understand how different server-side processing technologies can be used to collect and process the data from the form and how to go about using these technologies;
 8. Use the correct terminology when communicating about online questionnaires;
 9. Collect information about sources of help when developing online questionnaires in different contexts.
-

Introduction to online questionnaire production: Overview and options

Introduction

This page aims to offer an overview of the process of creating an online questionnaire and to provide an introduction to some of the key issues that must be considered when deciding on an option for implementing a questionnaire.

It will also focus on the different options available, and provide examples of the technical choices a number of researchers made to facilitate their studies, and their rationales for doing so.

Finally, it will outline some of the key skills required to successfully employ each of the different methods.

Overview of the process of creating an online questionnaire

A typical procedure for implementing an online questionnaire is as follows:

1. Create an HTML document with questions and form elements to collect responses.
2. Ensure that the document is correctly designed and formatted for consistency and accessibility.
3. Add server-side scripts to allow the information to be collected through emailing the results in a useable format or automatically populating a database or text file with them.
4. Add server and/or client side scripting to minimise measurement error and improving the quality of data collection. This may involve:
 - o Incorporating validation routines to check that the information submitted is in a suitable format and that no questions or selections have been accidentally missed.
 - o Establishing access control to (where needed) increase the chances that only those who are part of the sampling frame take part in the survey.
 - o Including routines to obtain information such as whether or not the respondent's machine has been used to submit the information previously, or how long it took to complete the form.
 - o Adding a facility to thank the respondent for their participation and assure them that their submission has been successful.
5. Add server and/or client side scripting to improve the respondent's experience of completing the questionnaire and thus to reduce non-response. This may involve:
 - o Preserving data entered so that it is not necessary to restart either the survey or a section of the survey if the respondent makes an error or is interrupted.
 - o Adding effective skip mechanisms or tailoring subsequent questions in the survey according to particular responses given.
 - o Providing a progress bar or similar mechanism to indicate roughly how far through the survey a respondent is.
6. Upload the document to a server to make it available online.

7. Carry out the necessary distribution activities by, for example, emailing members of a sample list and sending follow-up emails according to responses.

A researcher who wishes to carry out each of these stages independently is likely to need a wide range of web design and web-programming skills to do this effectively. Specialist software or institutional systems are also frequently used to automate or simplify the process.

Options for creating online questionnaires

There are three main options available to researchers who wish to implement an online questionnaire. They can either take advantage of **institutional systems** that may be available, use an **'off-the shelf' solutions** to host and/or produce and manage the questionnaire, or use an entirely **self-produced questionnaire**.

The choices made are likely to be influenced by issues such as the following:

The intended size and scope of the study

- The number of likely respondents;
- The length of time the questionnaire will need to be online for;
- Whether there is a need for more than one questionnaire to be running concurrently;
- Whether there is a need to link data from particular respondents over time;
- Whether the plan is to use the questionnaire in tandem with paper-based or other types of survey;
- If so, whether the facility is required to enter data from paper-based questionnaires into the same database as the online ones, or to use machine-readable paper questionnaires.

The level of resources and technical support available

- Whether a budget is available;
- Whether the researcher is working in an institution which provides technical support or tools for creating online questionnaires;
- Whether there is easy access to a server on which to house web-pages and scripts;
- Whether adequate administrative permissions are available to, for example, create and access a database on the server.

The technological requirements of the questionnaire

- Whether a simple replacement for a paper-based questionnaire is required, or whether there is a need for particular features such as the option to require a response to particular questions, the incorporation of audio and video materials, randomisation of questions, or the inclusion of branching or skip patterns

The distribution requirements

- The extent to which the researcher plans to control access to the questionnaire;
- Whether a particular sample group will be targeted;
- Whether a list of email addresses is available to enable contact with the sample;

- Whether the technological facilities are required to manage the scale of email correspondence by, for example, checking for submission and sending follow-up emails to particular groups.

Other requirements

- Whether particularly personal or sensitive information is to be collected that will need to be kept on a secure server;
- Whether there is a need to securely share access to the data with others;
- Whether there are particular style and layout needs;
- Whether there is a need for particular types of analysis;
- Whether there are any relevant institutional constraints such as the need for the questionnaire to be housed under the URL of the institution or for the questionnaire to meet institutional style guidelines.

The researcher's technical experience

- The level of experience of internet-mediated research and web design expertise;
- Whether the questionnaire has to be developed to a tight timescale, and the amount of time available to the researcher to develop skills such as the following:
 - HTML and CSS to produce and format pages;
 - Basic programming skills to allow freely-available scripts to be adapted and used;
 - More advanced programming skills to allow you to create your own scripts to meet your particular needs.

Institutional systems

For researchers working within an institution such as a university, a government department, an NHS trust, or a charitable organisation, technical support may be available which the researcher may be able to use to implement the questionnaire. In some, cases, support may extend to the bespoke conversion of a paper-based questionnaire to a web version.

Examples

1. The Association of American Geographers 'Internationalizing Geography in Higher Education' study

Michael Solem (Educational Affairs Director)

How did you implement the questionnaire?

The development process was relatively straightforward and the questionnaire was programmed by one of our technical staff.

Did you experience any problems at all?

We did experience some database issues during implementation (sometimes buttons didn't submit information properly), but nothing that couldn't be resolved fairly quickly.

Were there any issues around response rates?

We mailed a printed version of the questionnaire to everyone who didn't respond to the two online requests, which increased our response rate significantly. My experience has been that printed surveys are needed to motivate large numbers of non respondents to participate, who otherwise would ignore the Web survey.

Waverly Ray (Research Assistant)

What was the context of the research?

The research investigated factors that influence geography faculty members' participation in activities that serve to internationalize college and university campuses, including international collaborative teaching and research.

Why did you choose to carry out the study using an online questionnaire in tandem with a paper-based version rather than a paper-based version or an online version only?

To lower data collection costs, an online instrument was developed. The response rate from the online questionnaire was not as high as anticipated, so paper-based versions of the instrument were sent to online survey non-respondents.

Was this decision made before or after the development of the questionnaire?

The online questionnaire was developed first, followed by the paper-version of the survey. From the beginning of data collection, the paper-based version of the survey was made available online for respondents who for personal preference or accessibility reasons did not want to complete the online version.

Why did you choose to have the questionnaire produced and hosted in-house?

Staff and resources were available to produce and host the questionnaire in-house.

Did you consider any other options for implementing the questionnaire (e.g. off-the-shelf software and/or hosting solutions?) If so, why were they rejected?

To my knowledge, no other options for implementing the questionnaire were considered.

How involved were you in the technical implementation?

I was not involved in the technical implementation, other than as a trouble-shooter for the beta version of the online survey.

What did you actually have to do to get the questionnaire online (e.g. hand over a paper version / liaise over design and development / specify technical features required such as access control and format of resulting data)?

I provided the programmers with a paper version of the survey and the programmers took care of the design and development of the questionnaire. I did not provide the programmers with format requirements for the resulting data. As a result, I assigned numerical codes to the data prior to analysis. This step could have been eliminated if I had provided the programmers with format requirements.

Would you recommend your methods of implementing the questionnaire to other researchers? What were the advantages and disadvantages?

I would recommend my methods to other researchers, although I had the benefit of a staff of programmers that were responsible for the technical development of the online questionnaire which made an online questionnaire viable for my research. There were many advantages of an online questionnaire, including (a) the ease of contacting survey participants in far away places; (b) the low cost of using online methods versus paper/mail-based methods; (c) time for data-entry was lessened; and (d) errors resulting from data-entry were lessened, as well. The main disadvantage is that the anticipated response rate was higher than the actual response rate. So it was important that we had the resources to mail paper-versions of the questionnaire to survey non-respondents.

What advice would you give to people in working with technical support staff to get a questionnaire implemented?

In working with technical support staff, I would suggest that other questionnaires are reviewed so that there is a clear understanding of what format would best suit the aims of the research. Then, the finalized questionnaire items, the array of possible responses, and numerical codes for the resulting data should be provided to the technical support staff. Creating beta versions of the survey is important, so that the survey can undergo revisions.

Were there any issues that emerged through the process that you hadn't expected?

Technical issues arose that the support staff resolved without my involvement. The expected response rates were lower than the anticipated response rates, which resulted in delaying data analysis until paper-based versions of the questionnaire could be received from survey non-respondents.

What experience of web page production had you had prior to setting up these pages?

I had very little web page production prior to participating in this research.

Did you have to learn any new technical skills?

No, I did not have to learn any new technical skills.

2. Nicky Shaw (Lecturer in Operations Management, Leeds University Business School)

What was the context of the research?

We were a team of non-technical academics from social science and psychology disciplines. None had experience of setting up online questionnaires, though all were proficient in questionnaire design. The survey was of Leeds University academic staff (all grades) regarding work-life balance. The questionnaire comprised four sections (three research areas plus a demographic) and was quite long.

Why did you decide to use an online questionnaire?

Really, our driving force was cost (so the online bit was a relatively cheap option saving on photocopying and return postage) and ease of data entry. We were looking to have the responses migratable to a format compatible with SPSS 'by magic'!! Which worked! The questionnaire was in html and dropped all responses into a comma delineated format, which could be opened in Excel and copied across to SPSS.

When did you make the decision?

Actually, (as academics) we were more concerned with the questionnaire itself - which took around 1 year to develop - and the issues around making it available online were addressed proportionately quite late on.

Why did you use the university to host the questionnaire (as opposed to a commercial hosting service)?

Never occurred to us to pay outside people!! We had obtained a little funding, which was just enough to pay an internal computer guy within one of our departments plus incentives for recipients...

Were there many issues that emerged through the process that you hadn't expected?

All the points below came about as a result of conversation with other colleagues and the technical staff member involved in setting up the website. None were issues we had consciously considered prior to embarking upon an online questionnaire

- Password protecting the website, to prevent people completing the questionnaire from some other 'sample population'.
- We knew we wanted respondents to view the three research sections in a random order (to prevent fatigue on one particular section) and this was done automatically as people accessed the questionnaire
- We indicated the rough time duration for each section and colour coded the sections for clarity.
- To prevent people missing questions by mistake, a check was performed on missing answers for each section and the respondent informed of any omissions (highlighted in red). If they still wished to skip the question, the submit button would take them to the next page the second time around.
- Logic pathways had to be very clear for programming purposes (e.g. If answering X to question Y, go to section Z etc).
- A lot of attention had to be paid to the coding, both in SPSS and Excel, as well as the main data file and a few dry runs performed to make sure that data wouldn't be lost.
- The questionnaire was designed in Netscape and the layout appeared distorted initially when running in Explorer.

Where this kind of service is not available, there may also be standard procedures in place for implementing web forms once they have been created by the researcher. This may involve a mailing facility whereby the data is automatically formatted and delivered by email when the form is submitted. It may also involve a facility allowing the data to be downloaded in an appropriate format for importing into a statistical analysis, database or spreadsheet package.

Example

Tim Vorley (Department of Geography, University of Leicester)

Why did you use web-based questionnaires?

I used web based questionnaires not because they are innovative, but because they were wholly appropriate. They offered a dynamic alternative to paper based questionnaires, with the prospect of a higher return rate as there is no work involved other than the responses themselves. Also the automatic coding of answers meant that responses could be interpreted quickly and saved into databases and separate files relating to each respondent.

What experience of web page production had you had prior to setting up these pages?

I had no experience of web programming before creating the first version of the online questionnaire. HTML was easy to pick up and so I was able to teach myself the majority of what I needed to know, but support was available.

Did you use a WYSIWYG editor like Macromedia Dreamweaver or Microsoft FrontPage or code them by hand?

The web-form was created using a combination of WYSIWYG and hand coding – I think there are pros and cons to simply using WYSIWYG as it is important to understand what you are doing and how you are doing it to avoid confusion further down the line!

How did you go about implementing your questionnaire?

Firstly, I read up about basic HTML programming and learnt the basics. This allowed me to develop the skills needed to produce my questionnaire. I then took advantage of the University's facilities to host the questionnaire and to link it to server-side scripts allowing the data to be automatically emailed to me.

What kind of support did you receive?

The computer centre at the university offered a good level of support in setting up initial web-form and the server-side scripts for submitting results. Subsequently they helped iron out minor problems with the format of the questionnaire and the way in which results were received via server-based emails.

What exactly were these problems and how did you solve them?

Firstly I had problems with the frames in the webpage. By playing with the page, I managed to get the optimal viewing resolution lower so it appeared on even the lowest resolution and was easier to read. The problem with the automated replies was firstly with the coding. Initially the responses were uncoded and were received in a format that made it difficult to deal with the data. Both of these issues were resolved through trial and error and with the advice of the computer centre until I was receiving the results in the most user-friendly manner.

Why didn't you use off-the-shelf questionnaire software?

Using my limited and self taught knowledge of HTML programming I used Microsoft FrontPage to produce a clear and simple web-based form, designed using university corporate logos and colours. This was preferred over using off the shelf software because of the ability to build the web-form to my exact specification, also using widely available software meant teething problems were solved with the computer centre. At the initial time of conception I was also unaware of any appropriate software for designing questionnaires, and on further enquiry did not have the resources to purchase such software.

What did you do to check the data you received?

All of the respondents to the questionnaire were invited. With name and contact details on the questionnaire itself, responses were verified and constituted the basis for determining whether respondents would subsequently be interviewed. None of the data collected was explicitly presented within the research statistically or otherwise, but informed the interview process. The fact that the questionnaire was sent to invited respondents meant that security was not really an issue so I avoided many of the problems involved with ensuring the integrity of results and respondents.

Would you use the same system again if you were to do a similar study?

The system was easy to use and worked for me on this occasion. I am not the greatest fan of using surveys or questionnaires; however this was a lot easier to manage than a traditional paper based survey at every stage. The ability to modify questions and responses that did not work or were inappropriate after the pilot study was easy to do at any time when the initial questionnaire was set-up. If I did need to use a survey again I would use online over traditional every time!

What advice would you give to people in using systems like these?

I'd say to be persistent! If you have an idea, the chances are that it is possible but you just aren't sure how to do it at the moment! As a social scientist my forte is not designing websites or programming, but there are people out there who can help - the challenge is finding them and learning something new!

In some cases, the institution may have purchased a site licence for an off-the shelf survey creation and administration package which may be suitable for the purposes of the research project.

In educational institutions in particular, site-licences may also have been purchased for assessment software which may provide the facilities needed for the questionnaire. These are generally designed to allow students to answer a range of questions, such as multiple-choice questions or short text entry questions, online and submit their answers. The tutor can upload the questions and perhaps set passwords for each individual participant. They can then

download answers and see reports on the answers of individuals or groups. Because these are the same functions as those of basic survey creation software, this can be a straightforward (and free) way of implementing the questionnaire if the software is sophisticated enough for the needs of the study (e.g. by offering an adequate range of different question types). However, it should be remembered that these tools may only be available to participants within the institution.

The procedures for using these systems are often available on the institution intranet or through contacting computer services.

Using 'off the shelf' solutions

Software and services

A wide range of software is available that is designed to allow easy creation and administration of online questionnaires. There is a great deal of variety in the features available, and also the cost. There are a number of free open source examples which allow users to create basic forms with little knowledge of web programming. A variety of commercial software is also available, at a wide range of prices. Many of these offer different features according to the prices paid and it is common for them to offer free trials which are limited either in the number of respondents who can be surveyed or in the length of time the survey can be made available. The majority offer hosting services for those who do not have suitable access to a server, or who prefer to avoid installing and maintaining software.

The major advantage of these facilities is that they can reduce the need for web-design and programming skills. The majority of them offer a form-style interface into which the researcher adds questions and decides the type of web-form element they would like to use for answer. They also typically allow a range of formatting options and choices for validation, such as whether or not a response to particular questions will be required before submission. They can also provide server space if required and can allow a secure means of saving and retrieving data in a format such as comma-separated values (CSVs) suitable for easy input into common spreadsheet and statistical analysis applications. Other examples offer automatic analysis of data or facilities to easily cross-tabulate or filter results.

However, in some cases, they can also serve to limit the options available to researchers who may have particular design, validation or analysis needs not easily catered for by 'standard' software. Some basic knowledge of web design and programming can serve the researcher who wishes to 'tweak' the content or look of the surveys produced by these programs well.

See the 'Choosing software' and 'Using software' sections of this guide for further information, including the following:

- Examples of the different types of software available
- An overview of the different features and levels of sophistication commonly offered by providers.
- Guidelines for choosing software and the opportunity to develop a personalised checklist for use when comparing different options.
- A general outline on how to use the software.

Consultancy

There are many organisations that offer consultancy services for online questionnaire creation and administration. These range from advice and assistance with particular aspects of the process, to complete design, hosting, management and analysis services. Examples of these include those provided by academic research institutes such as the Bristol Online Survey services offered by the University of Bristol's Institute for Learning and Research Technology (<http://www.survey.bris.ac.uk>), and those provided by commercial companies such as the

survey shop, the research services arm of the snap survey software company (<http://www.snapsurveys.com/surveyshop/servicesweb.shtml>). Of commercial market research options, Evans and Mathur (2004)'s examination of the involvement in online surveys of the largest US-based and global market research firms provides an extensive list of the services offered as of late 2004.

Self-produced questionnaires

There are a number of cases in which a researcher may wish to carry out the whole process of creating and administering an online questionnaire:

- The researcher may have access to a suitable server on which to house a self-produced questionnaire, and have the web design or programming skills required to do this or be keen to control all aspects of the process.
- Technical support or institutional systems may not be available or may not be robust enough to provide an acceptable solution. (The use of an 'off-the-shelf' software solution may be more efficient in many cases).
- Where the technical requirements of the planned questionnaire are 'standard', it is frequently possible to obtain a wide range of freely available server and client-side scripts that can be used to set up the survey. With a little effort, a researcher with basic skills is likely to be able to acquire the knowledge to do this successfully and to customise where necessary. (Again, the use of 'off-the-shelf' software may be more appropriate).
- Where the nature of the planned survey is particularly complex and involves features that are not likely to be easily available without bespoke programming.

Example

Martin Bruder (School of Psychology, University of Cardiff)

1. Why did you decide to use a commercial internet hosting service rather than using the services offered by your university?

As I am doing my PhD at a number of different universities (Cambridge, Cardiff, Freiburg, Berlin), I did not want to depend on any university services for running my studies (I often find it hard to solve problems, when I am not actually there). I am now using a commercial provider (rather cheap) which offers php-support (I understand that universities often do not offer this support or need time to check any scripts). As long as no reaction time measures are involved, this seems to me a very good solution.

2. Why didn't you use some of the off-the-shelf questionnaire software which is available online?

I did not use any software package to produce my studies, because I like to have full control over what is displayed, how the data is stored, etc. (often, for example, software packages rely on cookies to temporarily store the data, which I don't like). At least for questionnaires and simple experiments, the programming seems to be manageable (and I am really not a computing specialist).

3. What kind of technologies did you use to implement your questionnaires, and have they been successful from a technical perspective?

My studies use nearly exclusively php (and mysql for saving the data) and have been working perfectly well. For questionnaire-based studies without reaction time measures, I think server-side programming really is the way to go and I am confident in saying that pretty much anyone with a Web browser can see and use my pages.

4. How did you achieve this level of accessibility?

JavaScript is only used for setting cookies (in order to check for multiple participation) and checking whether answers have been provided in order to remind participants that they might have forgotten to fill in an answer (both only in one of the studies). So if anyone has switched JavaScript or cookies off, he can still participate.

5. What did you do to check the data you received?

For fraud detection, I look at IP-addresses, cookies that I set and display a message if a cookie is set. Also, it seems a good idea to receive all the submissions by e-mail as well as saving them to a file. This way, one quickly gets a feel for "unusual" patterns of data submission (e.g. very many in a very short time) and can check the respective data sets more carefully.

6. What experience of web page production had you had prior to setting up these pages?

I had no experience before starting online studies (well, my first Website was actually a signup page for lab participants including a questionnaire measure).

7. Did you use a WYSIWYG editor like Macromedia Dreamweaver or Microsoft FrontPage, or code them by hand, and did you have to learn many new skills to do this?

I code my pages by hand using a freeware tool called HTMLkit. I had to learn to understand what html does (which is really fairly quick) and how pages can be formatted using Style Sheets. Learning this really does not take that long (maybe 15-20 hours).

8. Did you write your own php scripts to write to the database and send the email, or did you use or adapt an existing one? Again, did you have to learn many new skills?

I used and adapted existing scripts. Although I understand the basic principles of programming, I find it very hard to put it all exactly the right way and there are lots of scripts freely available online (many of the bits one needs for a questionnaire study - like setting cookies, retaining referrer URL, retaining IP-address are fairly standard and only short pieces of scripts). I have to say, though, that for one study that involves multiple randomisations and lots of different conditions, I had quite a bit of help by a friend who does computing. So if I had to do such a study all by myself, I might either have to put in more time to learn how to properly program or find a tool that I could use. But this does not apply to simple questionnaires.

9. Presumably you developed your questionnaires on a test server (apache?) on your home computer, and then uploaded them to the commercial provider (FTP?). Is this correct?

Yes, I installed foxserv (includes apache and mysql). Often, however, I upload it directly and test it online.

10. How did you choose which provider to go with?

There are lists of the 10 or so best Web providers for each country (my provider is in Germany). I knew I needed php- and mysql-support, so then I compared the prices for the ones that offered these and was either lucky or they are all good. So far, I never noticed any interruption of service.

11. Finally, would you recommend your methods to other researchers?

Yes, although, thinking about it, I might not have done things the most efficient way. The reason being that I did not mind so much investing some time - just out of curiosity.

Technical knowledge and skills required

Table 1. The importance of different sets of knowledge/skills when different methods are used to create an online questionnaire.

Knowledge/skills	Institutional systems	Self-produced	'Off the shelf' solutions
Knowledge of how to use questionnaire software	N/A	N/A	HIGH
Basic knowledge of HTML (and possibly CSS)	HIGH	HIGH	MEDIUM/LOW
Design issues	HIGH	HIGH	MEDIUM
Form elements	HIGH	HIGH	MEDIUM/LOW
Validation routines	HIGH	HIGH	LOW
Server issues	MEDIUM/LOW	HIGH	MEDIUM/LOW
Server-side processing	LOW	HIGH	LOW

Table 2: A list of the technical knowledge and skills required to produce an online questionnaire with the reasons why each skill is required.

Knowledge/skills	Reason
Understanding of the different facilities provided by different questionnaire software and hosting providers. Familiarity with the interface of chosen software and procedures for creating and managing the questionnaire.	There are a great many 'off-the-shelf' for online questionnaire software solutions available and if a researcher decides to take this route, time may be needed to become familiar with the options and facilities available. Once a choice is made, it is necessary to become familiar with how the software is used. The complexity varies greatly and options with relatively limited functionalities are often very quick and easy to learn. However, examples that offer advanced functionalities may also be relatively difficult to learn to use effectively.
Basic knowledge of HTML (and possibly CSS)	It will be necessary to create and appropriately format HTML pages and web forms. This can be done with a simple text editor or with 'What you see is what you get' editors such as FrontPage or dreamweaver. These can make the process of creating HTML pages more efficient, but a basic knowledge of HTML and CSS is very useful in using these packages effectively. Where form-creation software is used, knowledge of HTML and CSS will allow you to customise the look and feel of the questionnaire after it has been produced. Even with well-designed software, problems can emerge when HTML is automatically produced, and this knowledge will allow you to deal with this as effectively as possible by 'going to the source' if necessary.
Design issues	An awareness of design issues related to general web-design (use of colour, optimisation of graphics, design for a range of screen resolutions and browser types, accessibility issues etc) and those specifically connected to online survey design (progress bars, tables for matrices etc) will be needed to make the survey more effective and reduce measurement error. Where form-creation software is used, this allows easier customisation of templates and response to problems that the process of automatic HTML production may create.
Form elements	The use of common form elements will be required to collect the data for processing. Where software is used to automatically insert forms and elements, customisation will be easier.
Validation routines	Client-side validation routines using JavaScript can be used to check the data that has been entered before it is allowed to be submitted. If this is used effectively it can reduce problems of invalid data and multiple submission.
Server issues	The questionnaire will need to be placed on a suitable server connected to server-side processing facilities. Knowledge of server technology and security issues will be required to do this independently and, where support is available, a sound understanding will be helpful in getting this done effectively.
Server-side processing	The use of suitable server-side scripts will be necessary to allow the data to be processed and delivered or automatically inserted into a database. It will also be required to provide the respondent with a suitable acknowledgement that their submission has been successful.

Choosing software for online questionnaire production

Introduction

Online questionnaire software is designed to make the process of producing the HTML and necessary scripts possible for researchers with little or no technical knowledge. It generally provides a form-style interface to allow the questionnaire to be built up through inputting the questions and response types, and selecting pre-programmed features such as required responses, skip patterns, randomised ordering of choices, and resubmission checks.

There is a huge range of options available. At the time of writing, the WebSM searchable database of online questionnaire software and services (<http://www.websm.org/content.php?p1=82&p2=272&p3=1086&id=1086>) has 408 entries categorized according to the following:

1. Type (software or service)
2. Code availability of software (open or closed source)
3. Charges
4. Language
5. Country where offices are located

The Association for Survey Computing searchable software register of software also lists 79 examples of software that can generate online questionnaires. A simple *Google* search reveals many more examples of online questionnaire software with a vast range of features and prices.

This page will offer examples of the different types of software available and consider the different features and levels of sophistication commonly offered. It will also explore the factors involved in making a choice of software, allowing you to develop a checklist for use when comparing different options.

NB. Any references to prices or features are correct at the time of writing, but should be seen only as a snapshot of what is on offer in April 2005. References to particular software providers are given only as examples of the options available and should not be seen as an endorsement of any particular products or services, nor as a guarantee of quality.

Software plus hosting

For researchers without suitable access to a server, or for those who prefer to avoid installing and maintaining software, a wide range of hosting services are available. In the majority of cases, the whole process of producing and implementing the questionnaire is carried out online using a form-based interface. Many companies that offer 'software only' options also tend to have hosting facilities which registered users can take advantage of.

There are a great many companies offering services targeted at all levels of users from those offering design, hosting and analysis facilities for individual researchers for free, to those offering solutions to multinational companies for thousands of pounds. Many offer a variety of products targeted at different users with special rates for small-scale individual use. Others offer educational rates or even free services for researchers who are prepared to fulfil certain conditions such as having a link to the company on the institution website, and being prepared to contribute their survey questions to libraries of question types for reuse.

If the researcher decides that this type of service is suitable for his or her study, a number of questions must be considered when choosing a service. The aim should be to find an appropriate balance between cost and features required.

Cost

The cost of the services is often dependent on the number of respondents, questions and or surveys. Most charge a fixed monthly or annual fee, often setting limits on the number of completed surveys and charging additional fees for each survey beyond this maximum.

Many offer different levels of service, at different prices, usually with names such as 'basic edition', 'standard edition' or 'professional edition'. This generally involves differences in the maximum number of surveys allowed and differences in the levels of technical support offered. It may also involve additional features such as more advanced analysis and reporting tools, and more sophisticated question types and questionnaire options.

Given the competitiveness of the market, the majority of the services offer clear pricing information which can be linked directly from the home page. This tends to allow relatively easy comparison of different providers.

Available features

Many of the software company websites offer product demonstrations, tables of features, example surveys, and/or the creation of free demonstration surveys to 'sell their wares'. This makes it relatively easy to check the features that are available, but can also quickly lead to information overload.

A useful example survey which includes comments on the features illustrated by particular questions can be found on the demo pages of the SelectSurveyASP service at <http://www.classapps.com /SelectSurveyASPDemo.asp> (Note that this software does not offer a hosting service).

Surveyz is one of many options that offer a fully functional trial of their software and hosting service which allows the user to quickly get a feel for the functionality and options available. Registration is required to activate this trial and the trial is restricted to a one-month period, one questionnaire active at any one time, and a total of 25 responses.
<http://www.surveyz.com/>

It is a good idea to create a checklist of the features that you are likely to require. This can then be compared to the features offered by different providers. It can be particularly useful in enabling a minimum level to be established where a provider offers increasing levels of service at different prices.

The 'checklist questions' below will guide you through the process of developing such a checklist according to the needs of your study.

Examples of services available

The following examples are chosen as being representative of some of the different types of services available. In each case, a range of comparable options may be available.

Service	Comment
Bristol Online Surveys	Targeted at institutions requiring the option to have a number of different surveys and survey administrators. Highly customisable to the style needs of institutions including an option to run surveys on their server with an address that appears to be that of the institution.
Educara Survey	An open source tool which offers hosting for an annual fee. The price rises according to the type of use, with students paying the least, and commercial organisations the most.
Research together	Targeted at research students. Offers a simple questionnaire production and hosting service, allowing users to download results as comma-separated values for import into an analysis package. Relatively inexpensive one-off payment, but without many of the more sophisticated functionalities such as email list management or analysis tools.
SurveyConsole / QuestionPro	Both are divisions of the surveyanalytics company and they use the same software and interface, but with different pricing. May offer sponsored use for academic or not-for-profit projects if certain conditions are met. Also offer a range of free resources such as articles and question templates.
surveymonkey	Compares well with many of the other available services in terms of features, but is one of the cheapest commercial options.

surveywriter	Relatively expensive, but unusual in that charges are not made per period of use, but per completed survey and email invitation with a minimum of 200.
Surveyz!	A range of relatively sophisticated features. Targeted at individual researchers or at institutions. Offers academic pricing and free use for academic projects if certain conditions are met. Also has a range of resources such as articles on online questionnaires and copyable templates of questionnaires and questions.
websurveyor	Relatively expensive, but with a wide range of features. Offers both hosting and software only solutions.
zoomerang	Offers pricing for not-for-profit and educational institutions. Also offers a range of research services such as questionnaire administration, panel services and translation.

Software only

The use of software only solutions depends on the researcher having access to a suitable server with adequate administration permission to, for example, create and access a database. It also requires the software to be installed and configured correctly and for any possible problems that might emerge to be dealt with. Where these requirements can be met, this type of software can have a number of advantages. These may potentially include:

- Savings of cost, especially when used in the longer-term
- Increased control over data collection and storage
- More opportunities to work offline if required

Some examples of this type are also primarily designed as survey analysis software which can offer generally more sophisticated analysis tools where required.

Open source options

Locating software

Two of the major sources of information on open source software are SourceForge.net at <http://sourceforge.net> which is a repository of open source projects, and freshmeat.net at <http://freshmeat.net/> which is a listing of new software releases.

A simple search for 'survey' reveals a range of projects providing software for online questionnaire creation. These range from simple survey software with limited question types and functionality, to options with features which compare very well with the majority of commercial options.

Choosing options

When choosing whether or not to use an open source solution, it is important to consider the following potential problems:

1. They do not tend to offer hosting, and installation and configuration can be complex (though this may be less problematic than with commercial software-only alternatives).
2. The user-interface for the software can be more complex in some cases.
3. Documentation such as user-guides may not be as extensive.
4. Where problems are encountered, suitable technical support may not be guaranteed.

Making a successful choice depends on identifying whether or not the software provides the features required. The 'checklist questions' below will guide you through the process of developing a checklist according to the needs of your study which can be used when comparing software. In many cases the information on the features available is less extensive than commercial options, though many of the more established options offer demonstrations, examples and feature lists.

It is also necessary to ensure that the software can be installed and configured on a suitable server, and that someone with the necessary skills is available to maintain it successfully and deal with problems. An indication of what is typically involved in this is given by the administrator's manual for phpSurveyor at <http://phpsurveyor.sourceforge.net/docs.php>. Many of the other open source options also have similar installation guides which can be accessed through the project website or downloaded with the software.

Finally, it is important to check that the software has an adequate level of reliability. The JISC open source software advisory service, OSS Watch, at <http://www.oss-watch.ac.uk/> offers useful guidance on criteria for choosing reliable software. Some key factors identified are:

1. Reputation - Have you spoken to people with experience of a particular product?
2. Ongoing effort - Is there clear evidence of active development of the software?
3. Community support - Is there an active community of users on the project mailing list ready to answer questions from users experiencing problems?
4. Version - Has the latest stable version been available for some time and is there evidence that problems have been identified and fixed?
5. Documentation - Is this sufficient to allow you to decide whether or not the software is sufficiently well-developed for your purposes?

See the page of 'Top Tips For Selecting Open Source Software' at <http://www.oss-watch.ac.uk/resources/tips.xml> for further details.

Examples

The following are some of the main examples of open source software for online questionnaires (generally the more established and/or sophisticated options). A range of other options may be available.

Software	Comment
<u>Educara Survey</u>	In addition to the software, offers hosting for annual fees at rising prices for students, academic or not-for-profit institutions, and commercial organisations respectively.
<u>LE Survey</u>	Designed as a tool for running questionnaires as part of longitudinal studies. Allows respondents' responses to be matched to responses to previous questionnaires while maintaining confidentiality. In early stages of development at the time of writing.
<u>phpESP</u>	Well-established software with a working demo available allowing the features and user-interface to be tested.
<u>phpSurveyor</u>	A range of relatively sophisticated features. Well-established with useful documentation. Working demos are available allowing the features and user-interface to be tested.
<u>Mod Survey</u>	Well-established with sophisticated features such as dynamic content generation depending on previous answers. Requires the user to learn to use XML syntax particular to the software
<u>VTSurvey</u>	Easy to use and particularly useful for straightforward questionnaires as only the four main types of questions are supported (Multiple choice with radio buttons and Check boxes, and short and long text entry boxes).

Commercial options

Choosing options

Documentation and technical support for commercial options tends to be more extensive than that offered by open-source software, but in many cases there are charges for a support package. They frequently offer powerful statistical analysis, and many can be used to develop and analyse questionnaires off-line. In the majority of cases, hosting services are offered for an extra cost.

A number of the commercial options were originally designed as software to aid in the production and analysis of questionnaires in paper-based and other modes, often with online questionnaire functions added later and offered at extra cost. Others offer online questionnaire functions as part of a set of 'add-ons' allowing, for example, computer-assisted telephone interviewing (CATI) or surveys for Personal Digital Assistants (PDAs) or other mobile

computing technologies. This can make them a particularly effective choice when there is a need to integrate mixed-mode surveys.

It is useful to list the features required for your study which can be used when comparing software. The 'checklist questions' below will guide you through the process of doing this, providing you with some of the main questions to consider.

Examples

The following are some examples of commercial software for online questionnaires. A range of other options are available.

Software	Comment
<u>Questionmark Perception</u>	An educational assessment tool which offers many of the key features needed to create basic online questionnaires and has some of the more advanced features such as randomisation of questions and conditional branching. Potentially useful option if the software is available through the researcher's institution.
<u>SelectSurveyASP</u>	Relatively inexpensive. Offers 'classic' and 'advanced' versions with different levels of features at different prices. Has a working online demo and a useful example survey which includes comments on the features illustrated by particular questions. Offers a free installation service and free technical support.
<u>Snap Surveys</u>	Extensive options for mixed-mode surveys, offering a 'core product', Snap Professional with add-ons for questionnaires via internet and PDAs, and to allow scanning and multiple data entry. Expensive example of 'high-end' options.
<u>SphinxSurvey</u>	Extensive analysis tools including a version offering lexical analysis. Educational and public-sector pricing offered.
<u>StatPac</u>	The online questionnaire software does not include analysis tools, but it can be purchased alongside the statistics tools offered. Has basic statistical tools or an advanced version allowing multivariate statistical techniques. Technical support and updates are available free for three months, but are chargeable via annual support/maintenance agreements thereafter. A fully-functional version of the software can be downloaded and used for free, limited to 35 respondents for each survey. Download includes tutorials and extensive user guide.

Checklist questions

The following questions can be used as a checklist for comparing different products and services after deciding which of them apply to the questionnaire:

General

Does the service allow you to collect the maximum number of completed questionnaires you expect without extra charges?

Does it allow the maximum number of questions you need to include in your questionnaire?

Will you be within your budget if you use the service to keep the questionnaire open for the time you need?

Do you need only one questionnaire or do you need multiple questionnaires to be online at the same time?

Do you need a one-page questionnaire or one that spans multiple pages?

Do you expect all the participants to complete the questionnaire online or do you need to be able to use the online questionnaire in tandem with paper-based versions? Can paper-based versions of the questionnaire be easily produced?

Does the interface seem to be easy enough to understand and use in the time available?

Is online and/or telephone technical support on the use of the service available if required?

Question creation

Does it produce all the question types you would like to include?

Are there any limits to the number of choices for particular question types? Is this a problem for your questionnaire?

Can questionnaires be imported from word-processor or text files?

Is the layout and positioning of questions and answers acceptable? Can this be controlled?

Can you save the questions you have input and continue creating the questionnaire at a later date?

Is there the option of requiring a response to certain questions?

Can you randomise the order of possible responses?

Design and layout

Does it produce pages that have no major differences when viewed in different browsers or with different screen resolutions?

Is there a choice of templates? Are they suitable? Are the layout and colour schemes acceptable?

Do any templates allow easy customisation?

Is it possible to add extra design features to individual pages, such as images, extra text or line breaks?

Can you easily add features such as images, headers or footers to all pages?

Questionnaire settings

Is the statistical information offered by the software extensive enough for your needs? (e.g. can it record IP addresses of participants, approximate time taken, response rates for each question etc.)

Can the number of questions per page be easily controlled?

Can skip patterns be established so that the participant is directed to alternative questions depending on responses to particular questions?

Can the navigation be customised? Can the text on buttons be customised and is there a choice over whether or not a respondent can navigate back to completed sections?

Can multimedia stimuli such as video and animation be included?

Does it allow randomisation of the order of questions or responses within questions?

Can cookies be added to the participant's computer to prevent multiple submission from the same machine?

Can this use of cookies be controlled to allow particular machines to make multiple submissions (e.g. where a suite of computers may be used by different participants, or for data entry from paper-based versions)?

Does it allow for the inclusion and placement of a progress bar?

Implementation and security

Are the welcome pages and post-submission messages fully customisable?

Can you link to a webpage of your choice once the questionnaire has been submitted?

Can you enter a closing date for the questionnaire and add a customisable message to those who attempt to access it once closed?

Does it offer the access control you need, such as the ability to set passwords or limit access to those from particular IP addresses?

Are you satisfied by any security statements made?

Is there an option to encrypt the information on a secure server?

Distribution

Is there an email management list?

Can you insert individual email addresses?

Can multiple email addresses be imported?

Can you select the whole list or a subgroup of the list to send a message to?

Can you send personalised emails by merging personal information into a standard email?

Can the link to your questionnaire be easily incorporated into emails?

Can you send your entire questionnaire as an email?

Can you easily incorporate a link to the questionnaire into a website?

Can you record which of several links was selected by the participant?

Can a pop-up version of the questionnaire or of a link to the questionnaire be activated when a user enters the site?

Results and analysis

Can results be received by email alongside saving them to the database?

Does it allow the data to be downloaded in a format that can be easily imported into the statistical analysis package you plan to use?

Are there any limits to the number of times that you can download your data? Is this a problem for your study?

Can you choose whether you would like to include all data in the download, or only data from a particular period?

Can any extra information you need be easily included in the data files (e.g. IP addresses and start and completion times)?

Is the statistical analysis and reporting offered by the software sophisticated enough for your needs?

Can you view individual results and overall results on an on-going basis?

Does it offer the option to delete or edit results?

Are there the viewing options you require (e.g. Can you view the results as percentages or in the form of a particular type of chart)?

Can you easily access the statistical information you need from within the software?

Does it offer filtering options so that results can be analysed according to the response to particular questions (e.g. focusing on gender or age)?

Is cross-tabulation of results possible, so that correlations between particular questions can be analysed?

Using software for online questionnaire production

The interface and features of different online questionnaire software may vary. However, a typical procedure for creating and administering a questionnaire is shown below. At each stage, screenshots taken from different examples of software are provided to give an indication of how they tend to be used.

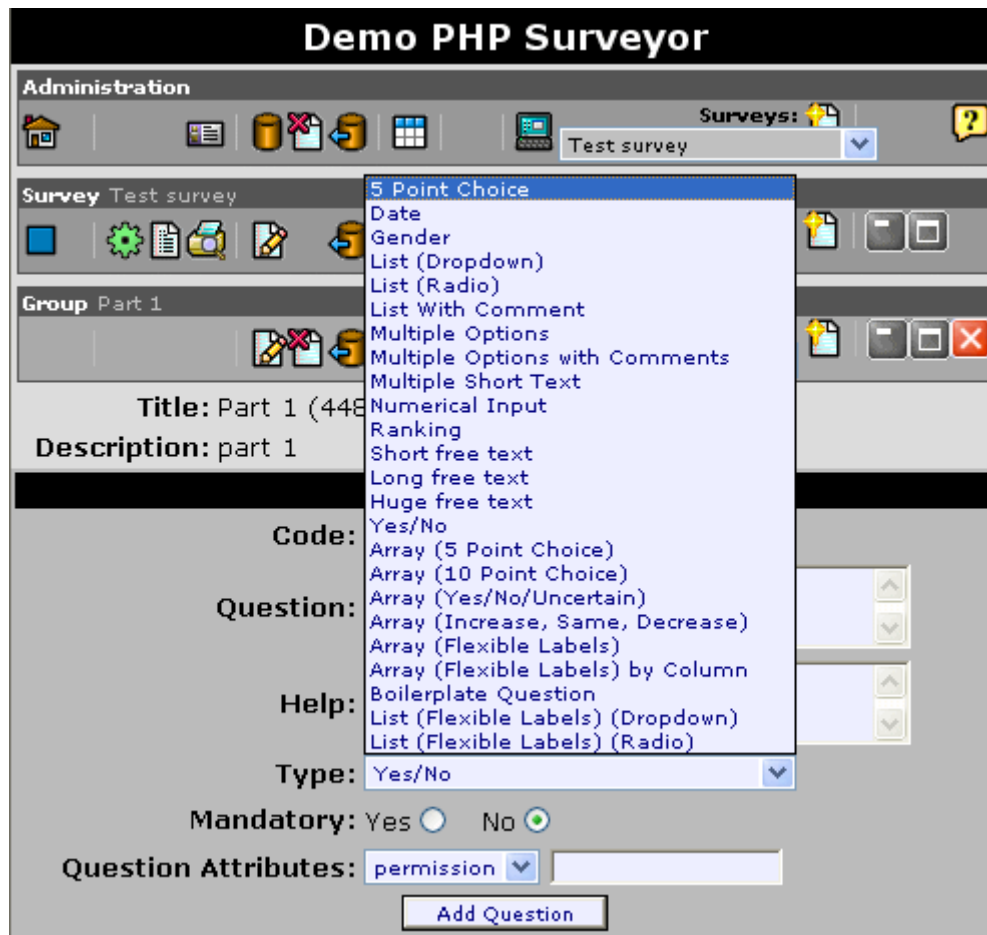
Create the questionnaire

1. Create a new page by adding a title, selecting formatting options and adding any logos required. (Alternatively, copy a previous survey to amend or choose a template from a list of options).



The survey editing interface of the SurveyMonkey.com service (<http://www.surveymonkey.com/>)

2. Add a question by selecting from a list of question types.



The phpSurveyor question creation interface (<http://phpsurveyor.sourceforge.net/index.php>)

3. Input the question and responses. Select options such as whether an answer is required, whether the order of responses should be randomised etc.

QuestionPro Online Survey Software Application - Microsoft Int... Step 2 of 3

Question Text

+ Font/Size

Answers

+ Font/Size

[Simple Mode](#)

1 [Text Box] Right

2 [Text Box] Right

3 [Text Box] Right

4 [Text Box] Right

Other Text Box Width 35 Height 0

Back **Preview**

The questionPro.com question editing interface (<http://www.questionpro.com/>)

4. Add any images, extra text or line breaks as required.
5. Add further questions on the same page or add further pages.


Edit the questionnaire settings

Many software plus hosting services offer a number of questionnaire settings options which allow the user to incorporate features such as the following:

1. Visual appearance or layout options such as customised headers, footers and images.
2. The inclusion of skip patterns if required, so that the participant is directed to alternative questions depending on responses to particular questions.
3. Customisation of button text, and choice over whether or not a respondent is given the option to navigate back to completed sections
4. Inclusion of multimedia stimuli.
5. Randomisation of the order of questions or responses within questions.
6. The addition of cookies to the participant's computer to prevent multiple submission from the same machine.
7. The option to automatically link to a particular web page on completion of the questionnaire, or to add a completion message.

8. The ability to enter a closing date for the questionnaire and to add a customisable message to those who attempt to access the closed survey.
9. The option of emailing the results to the researcher alongside saving them to the database.
10. Password protection for the survey.
11. The inclusion and placement of a progress bar.

The following is a screenshot showing some of the options provided by 'SurveyZ!'


The Ultimate Survey Solu
a Qualtrics company

[Help](#) - [Tutorials](#) -

My Surveys	Edit Survey	Distribute	View Results	Survey Controls
------------	-------------	------------	--------------	------------------------

Response Administration

Clear Survey Results Clear

! Please note that once cleared the data will not be able to be retrieved for analysis.

Administration Logs

Clear Results [View Logs](#)

Data Export [View Logs](#)

Controlled Survey Options

Survey

Password

Password Protect Survey ?

Save and Restore Options

Note : The Save and Continue option is **only valid for branched surveys**. In a branched survey respondents will be able to take the survey up till the branch and then continue from where they saved off.

Enable **Save and Continue** ?

Button Text :

Randomization

Randomize All Questions in The Survey ?

Randomly Select One Question ?

Question Display Mode

Single Question Display Mode ?

Automatic Confirmation/Thank You Email

Email A Copy of the Response to the Owner

Enable Confirmation Email ?

Subject:

Thank You for completing the survey

Survey Progress Bar Location

Specify the Location of the Progress Bar - Choose **None** if you do not want to display a Progress Bar

▼

Extract from the surveyz.com questionnaire settings page, named 'survey controls' (<http://www.surveyz.com/>).

Select the distribution options

Depending on your method of recruitment, many of the services allow you to easily choose and implement distribution options.

Email:

1. Insert or import the email addresses of your sample group.
2. Select the whole list or a subgroup of the list to send a message to.



The screenshot shows the SurveyMonkey.com interface for selecting message recipients. At the top, there is a navigation bar with links for Home, New Survey, My Surveys, List Management (highlighted), My Account, and Help Center. The date Tuesday, May 03, 2005 is displayed. The main heading is "Select Message Recipients" with navigation buttons for "<< Back" and "Next >>". Below the heading, a text block explains that before creating a message, recipients must be selected and that sending unsolicited email is prohibited. A dropdown menu for "Select email list:" is set to "Default List". Under "Select the recipients who will receive your message:", there are three radio button options: "Send message to all recipients", "Send message to recipients with status of" (selected), and "Send message to recipients where". The "Send message to recipients with status of" option has a dropdown menu open showing "not sent" (selected), "no response", "responded", and "declined". The "Send message to recipients where" option has a dropdown menu set to "email address" and a text field "starts with" followed by an empty input box.

The email management interface of the SurveyMonkey.com service (<http://www.surveymonkey.com/>)

3. Add the message, incorporating the link to your questionnaire and, where available, include fields to allow you to personalise the emails by merging individual information into a standard email.

Other options:

1. Copy the link to the questionnaire into a website (perhaps with the added functionality of recording which of several links was selected by the participant).



[Help](#) - [Tutorials](#) - .

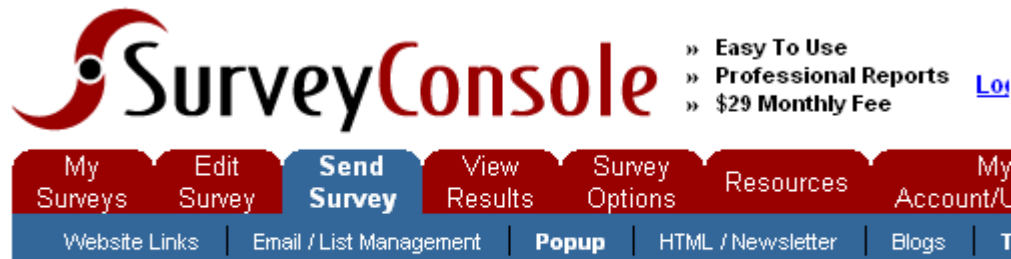
My Surveys	Edit Survey	Distribute	View Results	Survey Controls
Email	WebSite Link		Popup	Web

Print Surveys	
View Survey For Printout [Opens in a New Window] ?	Print Survey [Requires Javascript]

Text Link Surveys	
Link Type	HTML Code
Simple Text Link	<code> Link Text Here</code>
Text Link with Reference	<p>If you have multiple locations from where you will be linking to the survey and would like to track individuals clicking on each of the different sites use the <code>ext_ref</code> variable as a query parameter.</p> <p>The <code>ext_ref</code> variable is part of the Excel Download</p> <p>Example :</p> <pre> Text Here Text Here </pre>

The website link facility available at Surveyz.com (<http://www.surveyz.com/>)

- Alternatively, copy the necessary HTML and code into your website to activate a pop-up version of the questionnaire when a user enters the site.



Popup and Dynamic Surveys

Popup Survey (Every Time)

Popup Skip Count

Cut-And-Paste this code into your web page:

```
<SCRIPT
SRC="http://www.surveyconsole.com/console/JavaScript?
id=99370">
</SCRIPT>
```

The popup survey facility available at SurveyConsole.com (<http://www.surveyconsole.com/>)

Collect and analyse the results

Many software plus hosting services offer the opportunity to download results or to analyse results online through the package.

Downloading results

- Select from options allowing you to choose the type of data file to download (e.g. 'Comma-separated values' files allowing easy import into packages such as Microsoft Excel, HTML files, or SPSS Syntax files)
- Choose whether you would like to include all data in the download, or only data from a particular period.
- Decide on the format you would like to download the data in (e.g. whether all the data from an individual use should appear on one row, or whether multiple selections should appear on their own rows).

4. Select the extra information you would like to include in the data files (e.g. IP addresses and start and completion times).

The screenshot shows the 'Export Survey Data' interface for a survey named 'Test1'. At the top, there is a navigation bar with icons for Home, New Survey, Surveys, Libraries, Templates, Email Lists, and Reports. Below this, the main heading is 'Export Survey Data' with a help icon. The page title is 'Export Data for Survey 'Test1''. A descriptive paragraph states: 'This page exports data in CSV (Excel) format for the selected survey. Standard data is always exported for each report. To export, first select a data format. Then select the response and/or user data.' Under the 'Data Format' section, three radio buttons are present: 'User Responses' (selected), 'Individual Responses', and 'SPSS Format'. Under the 'Response Data' section, there are five checkboxes: 'Username', 'IP Address', 'Date Started' (checked), 'Date Completed', and 'Time Completed'.

Extract from the SelectSurveyASP data export page (<http://www.classapps.com/>)

5. Press the export or download button and save the file to a suitable location.
6. Open the file in your statistical analysis package.

Analysing results

Many service providers offer analysis facilities such as the following:

1. The ability to view individual results and overall results on an on-going basis.
2. Different viewing options (e.g. as percentages or in the form of various charts).
3. Basic statistics such as mean and standard deviation.
4. The chance to delete or edit results.
5. Cross-tabulation of results so that correlations between particular questions can be seen side-by-side and analysed.
6. Filtering options so that results can be analysed according to the response to particular questions (e.g. focusing on gender or age).

The following is a screenshot showing some of the options provided by 'SurveyZ!'

SURVEYZ! The Ultimate Survey Solution
a Qualtrics company

[Help](#) - [Tutorials](#) - [L](#)

My Surveys | Edit Survey | Distribute | **View Results** | Survey Controls

Data Results | Cross Tabulations | Export Data | **Advanced Reports** | S

Quick Reports

- [Cross Tabulate](#)
- [Excel/CSV Export](#)
- [Comprehensive Reports](#)
- [SPSS Command File](#)
- [Subgroup Criteria Analysis](#)
- [Response Editor](#) ?

Custom Reports

[Comprehensive Report](#) [Sur](#)

- [Pie Charts](#)
- [Line Charts](#)
- [Bar Charts](#)

Currently Active SubGroup Selection Criteria (Add/ Remove)

NONE

Excel / CSV ▾ Create Report

Previous Data Downloads and Exports

#	Timestamp	Record Count	File Type	Click Icon to Download	Download SPSS Command File	C

Extract from

the surveyz.com 'view results' page (<http://www.surveyz.com/>)

Software designed primarily for statistical analysis, but with online questionnaire capabilities also tends to offer more options such as multivariate statistical techniques or lexical analysis.

Introduction to HTML 1

Introduction

HTML (Hyper Text Markup Language) is the technical language that lies behind most web pages. Increasingly it is possible to get by without a great deal of knowledge of HTML by using WYSIWYG (What You See Is What You Get) software packages such as Macromedia Dreamweaver or Microsoft FrontPage. These tools allow you to create webpages and online questionnaires without any knowledge of HTML, but a basic knowledge remains useful as it will give you more control and allow you to deal with any problems that emerge more quickly.

This page will provide the knowledge required to produce an HTML page. Even if you intend to use a WYSIWYG package to create your online questionnaires it is still worth spending a short time developing a basic knowledge of HTML.

To produce a web page effectively, the only requirement is that you have a simple text editor, such as notepad for windows, and a browser such as MS Explorer in which you can test your pages. The next section explains how to use these tools, and it can be ignored if you are using a WYSIWYG editor.

Producing and editing web pages

The basic steps in producing and editing web pages are as follows:

Producing an HTML document

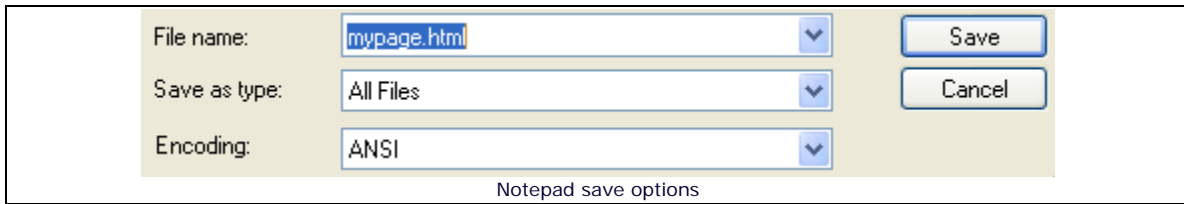
The first step in creating a web page is to produce a text file and save it as HTML.

To do this, you will need to open your text editor and copy the following basic web page into it:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>My web page</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body>
<h1>My Page</h1>
<p>Welcome to my web page</p>
</body>
</html>
```

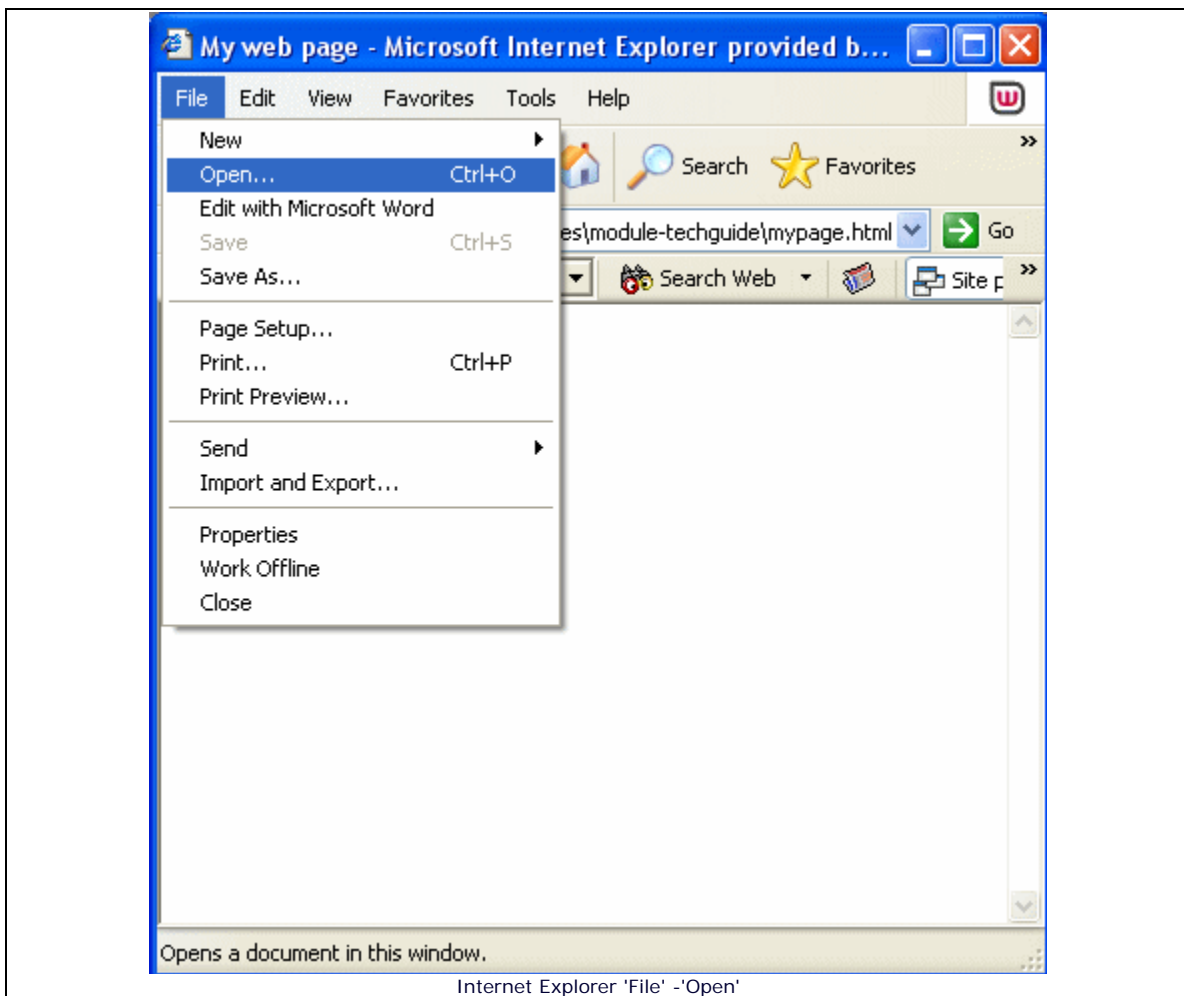
You then need to save the file as an html page by adding an '.html' or an '.htm' extension. Certain servers require a three-letter extension which means that it is probably safer to use the 'htm' extension, although in most cases, there is no difference between them. In notepad,

the save is done by choosing the options shown below. You should choose a folder to save the file in and remember where you saved this file.

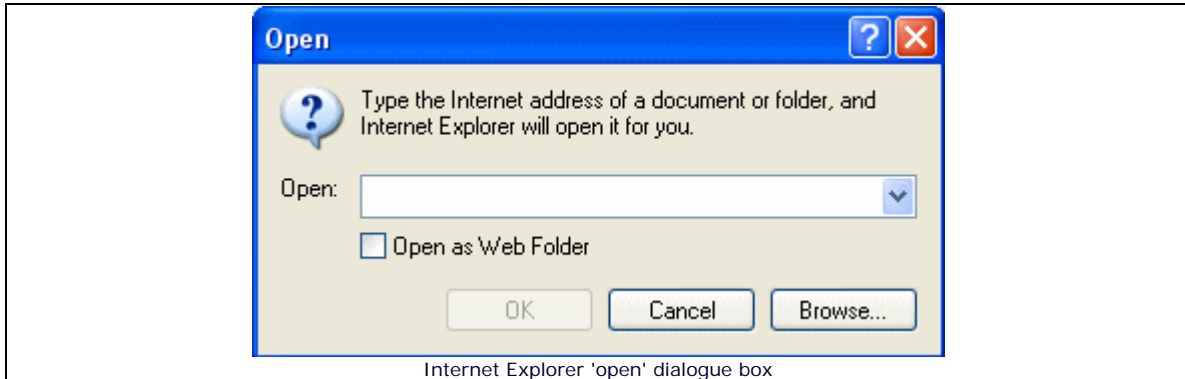


Viewing the document

To view this document, start your browser, and open the page by choosing 'File', then 'Open'. NB. Make sure you are working offline if you pay for your internet connection time. It is not necessary to be connected to view files saved on your computer (local files).



This will allow you to type the location and file name into the dialogue box (shown below), or select browse to navigate to the correct location.

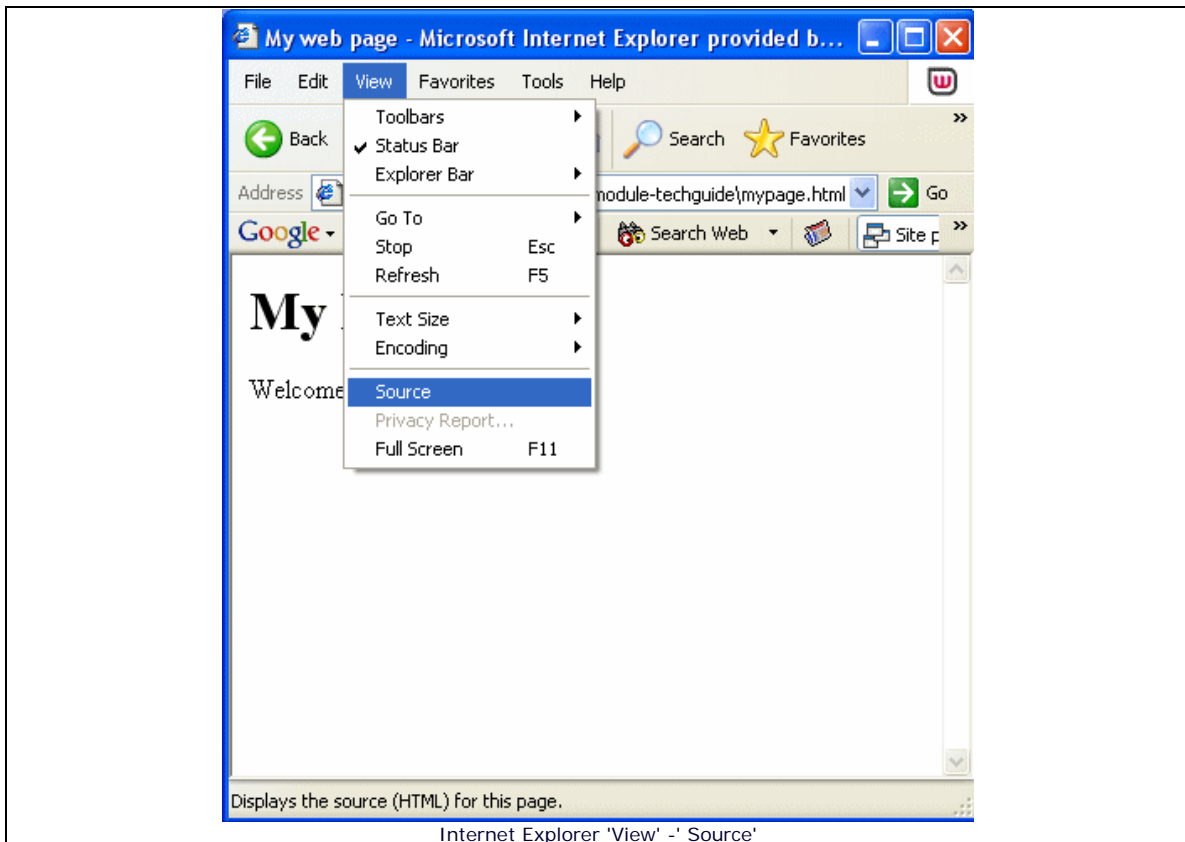


Changing the document

To make changes to the web page, you should change the source file in notepad, save the file, then refresh the page in your browser, by using the 'refresh' or 'reload' buttons (shown below).



You can open the source file of your own page to make changes by selecting 'View', then 'Source', as shown below.



You can also use this to view the HTML source of any web page on the internet. This is a very effective way of developing your knowledge of HTML, allowing you to view how particular effects within web pages have been created.

HTML Tags

HTML documents basically consist of text marked with tags which tell the browser how to present its layout and style.

These tags can consist of elements, attributes and values.

e.g.

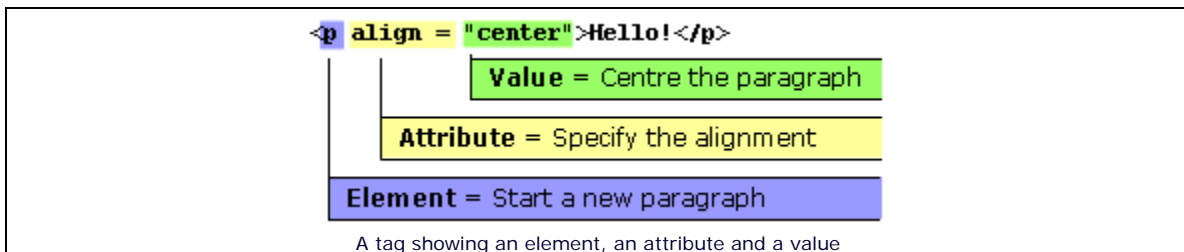
The following tag:

```
<p align="center">Hello!</p>
```

produces the following when placed in an HTML document:

Hello!

The tag is made up of an element, an attribute and a value as follows:



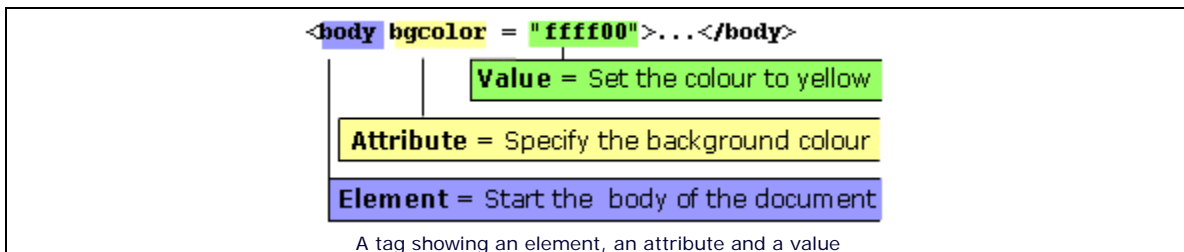
Similarly, the following tag:

```
<body bgcolor="#ffff00">...the body of the HTML page goes here...</body>
```

adds the following yellow background colour to the entire page:



This is also made up of an element, an attribute and a value as follows:



Further details about these tags are given below.

Elements

The usual pattern for elements tags is:

<Open tag> text </close tag>

e.g. 1

```
<p>To everyone:</p><p>Hello!</p>
```

This produces two new paragraphs. The first consisting of the words 'To everyone:' and the second consisting of the word 'Hello!', as follows:

To everyone:

Hello!

e.g. 2

```
<p>To everyone: <strong>Hello!</strong> </p>
```

This produces:

To everyone: **Hello!**

e.g. 3

```
<p>To everyone: <strong>Hello!</strong></p>  
<p><em>Hello!!</em></p>
```

This produces:

To everyone: **Hello!**

Hello!!

It is good practice to ensure that your tagging is symmetrical as in the following diagram:

```
<p><em><strong>Hello everyone!</strong></em></p>
```

A symmetrical tag structure

and to avoid an unsymmetrical structure as in the following:

```
<p><em><strong>Hello everyone!</em></p></strong>
```

An unsymmetrical tag structure

This will ensure your page comply with standards help to avoid the chance of information from being displayed incorrectly.

Although most tags follow the open tag (<>) and close tag(</>) format, there are certain key tags which stand alone.

The three most common examples are:

```
<br>
```

which produces a new line.

```
<hr>
```

which produces a horizontal line, e.g.



```
<img src ="smiley.gif">
```

which shows an image (on this occasion an image called 'smiley.gif' saved in the same folder as the web page).

e.g.



It is now increasingly common to see these tags 'closed' by the addition of a space and a forward slash before the closing bracket, as follows:

```
<br />
<hr />
<img src ="smiley.gif" />
```

This is because the latest standards of XHTML state that all tags should be closed.

Attributes and values

Depending on the element used, different attributes and values can be applied. Some require an attribute and value while, for others, they are optional extras.

The typical pattern is to include them within the tag as follows:

```
<element attribute1="value" attribute2="value">Add content
here...</element>
```

Try to maintain consistency in the use of spaces and inverted commas as in the pattern above. This will ensure that browsers render the HTML fully and consistently, and it will also ensure your code meets the latest standards.

Thus, for example, the following should be avoided as the browser may not connect the value to the attribute correctly:

```
attribute1 = "value" attribute2 = "value"
attribute1 = value attribute2 = value
attribute1="value"attribute2="value"
```

Typically, attributes and values alter the style of text, images or other parts of a page. For example, they change the size, colour, or layout in different ways.

The effect of different attributes and values is discussed within each section below.

NB. For size values, it is good practice to use relative sizing rather than absolute sizing. Relative sizes are expressed in percentage terms or, in the case of font-size, in terms of 'ems'. These sizes allow the user to change the appearance of the page according to preference or accessibility requirements. The use of absolute values such as pixels prevents resizing in many browsers which may cause accessibility problems for users of the questionnaire. This issue is discussed in greater detail in the 'Key design issues' section of this technical guide.

HTML Document structure

HTML documents should begin with a DOCTYPE (document type) definition, known as a DTD. This declares what type of page it is and what language is being used, and it allows the page to be validated as conforming to Worldwide Web Consortium (W3C) standards.

The breakdown of the DOCTYPE information is as follows:

- DOCTYPE HTML PUBLIC - Declares the page as an HTML page for public browsers.
- HTML 4.01 Transitional - Declares that HTML version 4 is being used.
- EN - Declares that the page is written in English.

This must be followed by a root element '**<html>**' which allows the browser to display the page. The **<html>** tag must be closed at the **very end** of the document using '**</html>**'

Following this information, HTML documents are basically divided into two sections. The head and the body.

The beginning of each is marked within the 'open tags' **<>** and the end of each is marked by the close tag **</>**. They are highlighted with arrows (▶) in the HTML page below:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  
<html>  
▶ <head>  
<title>My web page</title>  
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">  
▶ </head>  
▶ <body>  
<h1>My Page</h1>  
<p>Welcome to my web page</p>  
▶ </body>  
</html>
```

More details about these sections are given below.

The head

The head contains information which is basically not intended for display. It is loaded into the browser before the body section.

A typical head sections is as follows:

```
<head>

<title>Add pagetitle here</title>

<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<meta name="description" content="add description here">
<meta name="keywords" content="keyword1, keyword2 etc.">
<meta name="author" content="author name">
<meta name="copyright" content="copyright information ">

<link href="filename.css" rel="stylesheet" type="text/css">

<script language="javascript" src="filename.js"
type="text/javascript"></script>

</head>
```

Each section of this is explained below:

<title></title>

```
<title>Add pagetitle here</title>
```

This states what the title of the page is, allows the title to be displayed at the top of the browser window, and provides a title that can be saved to a user's favourites list.

meta commands

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<meta name="description" content="add description here">
<meta name="keywords" content="keyword1, keyword2 etc.">
<meta name="author" content="author name">
<meta name="copyright" content="copyright information ">
```

These commands provide information about your pages. They are useful for search engines which use them to store information about pages for searching. They also provide useful information for allowing pages to be stored in repositories.

A common meta command is:

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
```

This describes the type of content and which set of characters is in use.

Other meta tags (mostly self-explanatory) are as follows:

```
<meta name="description" content="add description here">
<meta name="keywords" content="keyword1, keyword2 etc.">
<meta name="author" content="author name">
<meta name="copyright" content="copyright information ">
```

Particularly useful meta tags for online questionnaires are as follows:

```
<meta name="HTdig" content="htdig-noindex">
<meta name="robots" content="noindex">
```

These tags prevent search engines from indexing the page and thus prevent the page from being included in searches. (Note that some search engine robots may ignore these tags and index the page, but use of the tags is still likely to reduce indexing).

Cascading Style Sheets (CSS)

```
<link href="filename.css" rel="stylesheet" type="text/css">
```

Cascading Style Sheets are useful for applying styles (layout, colours, positioning etc) across all elements of web pages. The aim is to separate the presentation of web content from the structure.

The Style Sheet information is placed as follows:

1. In the document head between the tags

```
<style type="text/css">
```

and

```
</style>
```

or

2. In a separate file which the web page refers to through a link placed in the head (This allows changes made to one file to work across all the pages which link to that file and prevents the need to change each file individually). The link is as follows:

```
<link href="filename.css" rel="stylesheet" type="text/css">
```

See the 'Introduction to CSS' section for more information on the use of Style Sheets.

Scripts

```
<script language="javascript" src="filename.js"
type="text/javascript"></script>
```

Scripts (e.g. JavaScript or VB Script) are placed in the head. A reference to the script is then placed in the body which 'calls' the script in order to perform an action such as checking that a form element has been completed. The script will be loaded into the browser before it is needed as the head section loads before the body.

Scripts are placed between the tags

```
<script language="javascript" type="text/javascript">
```

and

```
</script>
```

As with CSS, scripts can be saved in a separate file which the web page refers to through a link placed in the head (Again, this allows changes to be easily applied to all the pages which use a particular script).

The link is as follows:

```
<script language="javascript" src="filename.js"
type="text/javascript"></script>
```

See the 'Introduction to JavaScript', 'Key design issues' and 'Form validation' sections for more information about scripting.

The body

The body of an HTML document contains the main display content. It is here that text, images, links, form elements, tables and lists are placed.

A brief overview of each of the main types of content (with the exception of forms) is given in the next section of this guide, 'Introduction to HTML 2'. For information about creating forms in HTML, see the 'web forms' section.

Introduction to HTML 2

Introduction

The body of an HTML document contains the main display content. It is here that text, images, links, form elements, tables and lists are placed. The following paragraphs give a brief overview of each of the main types of content, with the exception of forms which are covered in the 'Web forms' section of this guide.

In each case, general information about controlling style and layout within tags is included. However, it is a good idea to consider the use of Cascading Style Sheets to create these effects (See the 'Introduction to CSS' section of this guide). This tends to lead to 'cleaner' HTML with less need for repetition of long tags with a great deal of attributes and values for style. It also makes it easier to alter styles across an entire page or series of pages and increases accessibility.

Text

Text elements

There are a range of elements used to format text. The main ones are headers, paragraphs and line breaks.

There are six levels of headers, marked by a tag with an 'h' followed by a number from 1 (the largest) to 6 (the smallest).

e.g.

```
<h1>This is my main header</h1>
```

Text can be positioned on a new line by inserting a line-break (
) or on a new paragraph, by enclosing it in the <p>...</p> tags.

e.g.

```
<p>This is the first paragraph</p>
<p>This is the second paragraph which includes a<br />line break
in the middles of the line.</p>
```

Additionally, there are a number of styling tags which can, among other effects, underline, italicise or embolden text. Some of these are:

Tag	Effect	Comment
...	Adds <i>emphasis</i>	(Usually renders in a browser as italicised text, but generally preferred to the italicise tag <i>...</i> as it allows the effect to be adapted in specific cases where, for example, the user is using a text-to-speech browser).
...	Makes text stronger	As above. Usually renders as bold but generally preferred to the ... tag.
<df>...</df>	Indicates a definition. <i>Usually renders as italics.</i>	Other 'Logical styles' that indicate a particular effect that the browser may interpret in different ways include: <cite>...</cite> for titles, <code>...</code> to show sections of computer code, and <kbd>...</kbd> to represent typed text (usually renders as mono-spaced 'typewriter' style text).
<u>...</u>	<u>Underlines</u> text	Care is needed with the use of this tag as underlined text can frequently be confused with hyperlinks.
^{...}	Places text ^{above} the ^{horizontal} line (e.g. for footnote numbering)	The _{...} tag can also be used to create the ^{opposite} effect.

<code>...</code>	Produces a line through the text (strikethrough) .
---	---

Text attributes and values

Alignment

Paragraphs of text can be aligned using the 'align' attribute as follows:

```
<p align="center">Hello!</p>
```

produces the following when placed in an HTML document:

Hello!

```
<p align="right">Hello!</p>
```

produces the following when placed in an HTML document:

Hello!

...

For other formatting, the ... tag is required. This takes attributes that affect the chosen typeface, size and colour, as follows:

Attribute	Possible values	Comment
<code></code>	1, 2, 3, 4, 5, 6, 7 +1, +2, +3, +4, +5, +6, +7 -1, -2, -3, -4, -5, -6, -7	The default size for paragraph text is 3, with higher numbers producing larger text. The default size can also be increased or decreased by a certain amount by adding or subtracting by a value from 1 to 7.
<code>font face="value"</code>	Any font, or 'font family' (see comment).	It is important to remember that the user's computer may not have particular fonts that you may wish to use. Using common fonts is recommended, as is the use of 'font-families' which provide the browser with information about which fonts should be used as a replacement in a case where a particular font is not available. Thus the use of the tag <code></code> tells the browser to use Arial if Verdana is not available, followed by Helvetica, and finally by the default sans-serif font.
<code>font-color="value"</code>	A hash-mark (#), followed by a six-figure 'hexadecimal' colour code. e.g. <code>#FF0000</code> = red	There are 216 colours in the 'web-safe' colour palette. These colours are recommended as they are not subject to variation on different types of monitors and systems. The resources section contains a link to a palette of web-safe colours organised by either hue (colour) or value (lightness). This makes it easier to design appropriate colour schemes, using these colours.

<body> attributes and values

Attributes and values can also be added to the body tag to set default text and link colours and background images and colours for the entire document.

e.g.

A background colour can be applied to the whole document by adding the following:

```
<body bgcolor="#00ffff">...the body of the HTML page goes here...</body>
```


This adds the following blue background colour to the entire page:



The colour is added through the use of a six-figure 'hexadecimal' colour code (in this case '#00ffff'). The code is preceded by a hash mark (#). A page containing the codes for the 216 'web-safe' colours can be found in the 'Further resources' section of this module.

Alternatively, a background image can be used as follows.

```
<body background="bluewhitebg.gif">...</body>
```

This 'tiles' the image across the screen, repeating it vertically and horizontally to produce a background. The image () is called "bluewhitebg.gif" and is saved in the same folder as the document. The effect is as follows:



Care must be taken when using such background images as they may not render well on the screen. If text is placed on top of backgrounds created using such images, this may also make the page difficult to read and affect its accessibility.

Text and link colours can be set using the following attributes with different values:

```
<body text="#000000" link="#000066" vlink="#660066"
alink="#ff6600">...</body>
```

This will apply the following colour options to the whole document:

text will be black (#000000).

links will be blue (#000066).

visited links will be purple (#660066).

active links will be orange (#ff6600).

Links

Linking text

The basic tag for creation of links is:

```
<a href="http://www.le.ac.uk">Click to go to the University of
Leicester homepage</a>
```

This produces a link as follows:

```
Click to go to the University of Leicester homepage
```

It is possible to change the link through adjusting the position on the tags:

```
<p>Click to go to <a href="http://www.le.ac.uk">the University of
Leicester</a> homepage</p>
```

produces:

```
Click to go to the University of Leicester homepage
```

It is important to avoid enclosing other tags such as paragraph tags within the link tags as this is likely to prevent the link from working correctly.

A basic link will open in the same window. To open the link in a new window, **target="_blank"** is added to the link:

```
<p>Click to go to open <a href="http://www.le.ac.uk"
target="_blank">the University of Leicester</a> homepage in a new
window</p>
```

produces:

Click to go to open the [University of Leicester](http://www.le.ac.uk) homepage in a new window

Where these types of links are used, it is good practice to inform the user that the link opens in a new window and to ensure that such links are used consistently throughout. This is likely to reduce the usability problems that can occur when users are unaware that a new window has been opened and are thus confused by the fact the 'back' button appears to have been deactivated.

Linking images

Images can also be used as links:

```
<a href="http://www.le.ac.uk"></a>
```

produces:



Linking from an image automatically adds a border around the image. To remove this, **border="0"** is added to the link. As with all images, it is also necessary to add 'alt' text to provide an alternative description for text-only browsers or in any situation when the image cannot be displayed.

```
<a href="http://www.le.ac.uk"></a>
```

produces the following:



Email links and links within pages

It is also possible to add a **mailto** link which automatically opens the user's email program (if available) with the correct address automatically filled in:

```
<p>Our email address is <a
href="mailto:OnlineRM@le.ac.uk">OnlineRM@le.ac.uk</a></p>
```

produces:

Our email address is OnlineRM@le.ac.uk

N.B. It is a good idea to refer to the email address in full in the link rather than using a link such as [email us](#) as this allows users who do not have or use an email program that can be automatically activated to see and copy the address easily.

A final type of link that can be used is a link within a page. This is done by inserting anchors in the page at the point you wish to link to. The anchors are named using `` tags. The following tags produce anchor links to the headings 'Anchor 1' and 'Anchor 2'.

```
<h2><a name="Anchor1"></a> Anchor 1</h2>
<p>&nbsp;</p>
<h2><a name="Anchor2"></a> Anchor 2</h2>
```

This produces:

Anchor 1

Anchor 2

To link to these anchors, the following code is used so that when the link is clicked, the user is taken to the appropriate anchor point.

```
<p>Link to <a href="#Anchor1">Anchor 1</a></p>
<p>&nbsp;</p>
<p>Link to <a href="#Anchor2">Anchor 2</a></p>
```

This produces the following links. Their effect can be seen by selecting them.

Link to [Anchor 1](#)

Link to [Anchor 2](#)

Relative and absolute linking

Links between pages are of fundamental importance in the creation of a site. Absolute linking involves the inclusion of the entire URL as it would appear in the browser (e.g. <http://www.le.ac.uk/etc/etc.htm>). This would be the typical way of linking to external sites.

However, the usual method of linking pages of the same site and located on the same server is through the use of relative links. This section explains how these links work.

If all the site files are located in the same folder:



they are linked simply by adding the name of the document to be linked to.

e.g. For a link on the homepage (index.htm) to 'page1.htm', it is simply necessary to include the page name and extension in the link:

```
<a href="page1.htm">Page 1</a>
```

If the folder was placed on the server for the URL 'http://www.le.ac.uk', the link would take the user to the following URL:

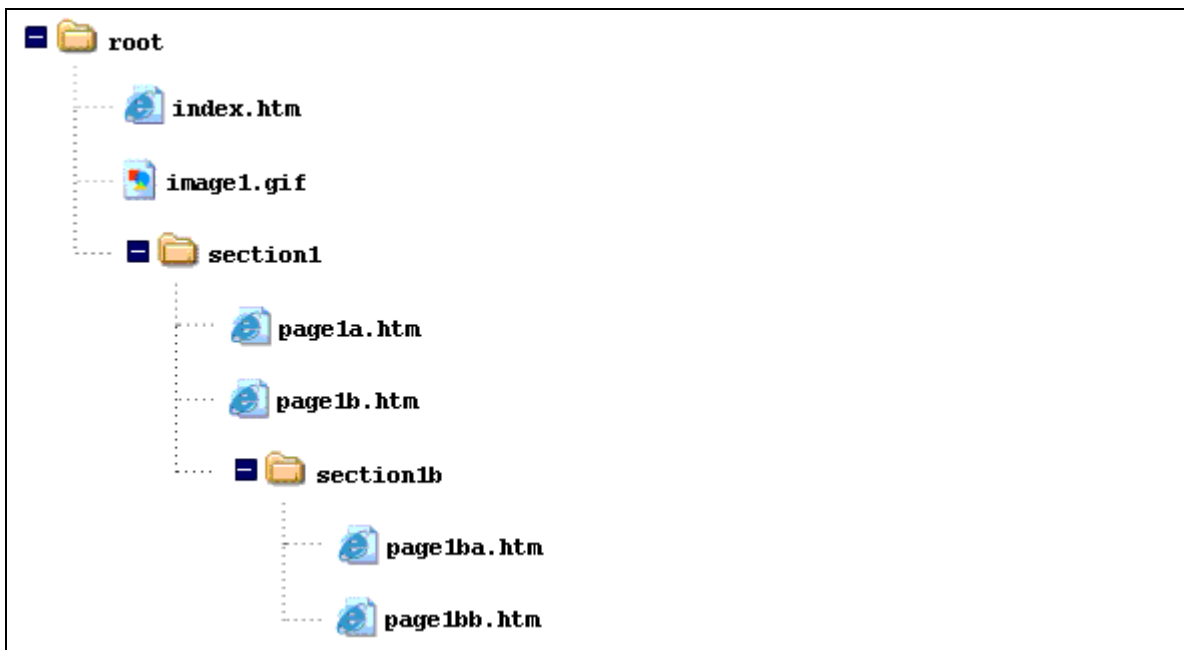
```
http://www.le.ac.uk/page1.htm
```

Similarly, if 'page2.htm' includes an image called 'image1.gif' it is simply necessary to link to the image through its name and extension.

```

```

However, if the page is held in a folder':



it is linked by including the folder name.

e.g. A link on the homepage (index.htm) to 'page1a.htm'. :

```
<a href="section1/page1a.htm">Page 1a</a>
```

The folder name must be included with a forward slash

e.g. 2. A link on the homepage (index.htm) to 'page1ba.htm':

```
<a href="section1/section1b/page1ba.htm">Page 1ba</a>
```

Both folder names in the path are included.

If the folder structure was then placed on the server for the URL 'http://www.le.ac.uk', the link would take the user to the following URL:

```
http://www.le.ac.uk/section1/section1b/page1ba.htm
```

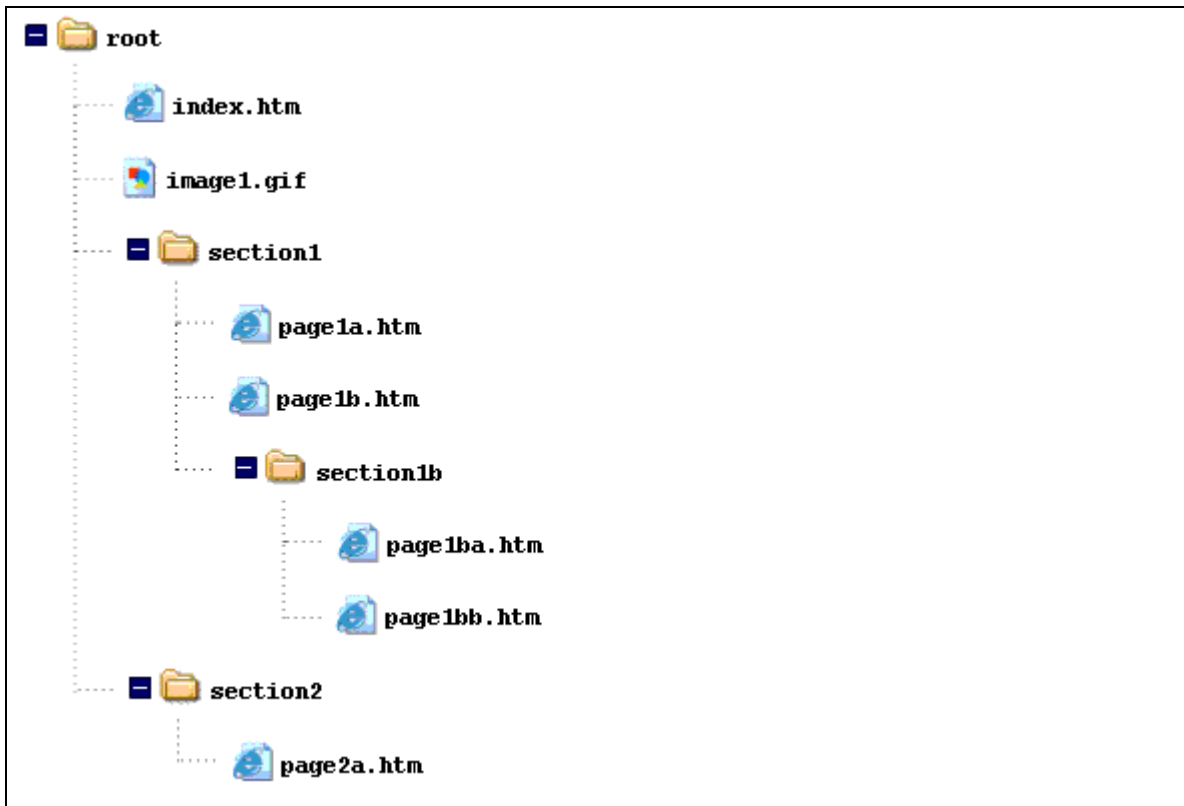
For a link to a page further up the file tree, '..../' is added to the link.

e.g. for a link from 'page1ba.htm' to 'index.htm', the link would be as follows:

```
<a href="../../index.htm">Home page</a>
```

Using this notation in combination, it is possible to link to any file in the same site.

Consider the following folder structure:



A link from 'page1ba.htm' to 'page2a.htm' would be as follows:

```
<a href="../../section2/page2a.htm">Page2a</a>
```

i.e. open the page called 'page2a.htm' in the folder called 'section2' in the folder two levels above this one ('section1b').

and a link from 'page2a.htm' to 'page1ba.htm' would be:

```
<a href="../section1/section1b/page1ba.htm">Page1ba</a>
```

i.e. open the page called 'page1ba.htm' in the folder called 'section1b' in the folder called 'section1' in the folder one level above this one ('section2').

Images

Formats

Though others are available, the two most common types of images in use on the internet remain the Graphic Interchange Format (GIF) and Joint Photographic Experts Group formats (JPEG). The filename extension for a gif is **imagefilename.gif** and for a JPEG, **imagefilename.jpg**.

In general terms, the use of gifs is more appropriate for graphics with a limited number of colours or line drawings. Jpegs are suitable for images with a greater number of colours such as photographs.

In either case, it is essential that the correct extension is used for the image to work.

Attributes and values

As seen in the links section above, the basic tag for the insertion of an image is as follows:

```

```

The image source (src) value is the path to the file name which can be expressed in absolute or relative terms (see the links section).

In addition to this, the following attributes and values are available for use. Some should be always be included, while others are optional.

Attribute	Possible values	Comment
width="value" height="value"	a value in pixels. e.g. width="50" height="50"	Although an image that is included without the height and width attributes will display at its original size, it is good practice to always include them. This allows the browser to load the information more effectively and to layout the page correctly even before the image loads. This is helpful for slow connections.
alt="value"	A description of the purpose or function of the image to allow its significance to be understood by users who have text-only browsers or in any situation when the image cannot be displayed.	To make pages accessible, all images should have an alt description. However, it is not necessary to describe every detail. The text should give as concise an indication of the image's function as possible. Where longer descriptions are required, these should be provided through links. Images with no importance such as spacing images should be given an empty alt tag "" to make it clear that this is the case. Any images used for buttons, icons, logos etc, should provide the information in the clearest and most concise way possible. e.g. "*", "[?]", "[!]"
border="value"	A value in pixels. e.g. border="2"	This is most commonly used to remove the border automatically placed around images used as links, by adding a value of "0".

align="value"	left, right top, middle, bottom e.g. align="top"	To centre an image it is necessary to place it within a centered paragraph using the <p></p> tags. This can also be used to align left or right. The top, middle and bottom values align the image on the horizontal line.
hspace="value" vspace="value"	A value in pixels e.g. vspace="10" hspace="10"	Sets the space around the top and bottom edges (vspace) and left and right edges (hspace) of an image. ▶ This graphic has an hspace of 10 pixels. ▶ This has an hspace of 30 pixels.

Example

The following full image tag:

```
<p align="center">TextText.</p>
```

produces the following:



Lists

List elements

The creation of lists in web pages is straightforward. It basically involves the following tag pattern:

<type of list>

<list item>First list item**</list item>**

<list item>Second list item**</list item>**

<list item>Third list item**</list item>**

</type of list>

The three main types of list are shown in the table with the HTML that produced them.

An unordered list (bullets) is as follows: <ul style="list-style-type: none">• Item 1• Item 2• Item 3	An ordered list (numbered) is as follows: <ol style="list-style-type: none">1. Item 12. Item 23. Item 3	A definition list is as follows: Term 1 Definition 1 Term 2 Definition 2 Term 3 Definition 3
<pre> item 1 Item 2 Item 3 </pre>	<pre> item 1 Item 2 Item 3 </pre>	<pre><dl> <dt>Term 1</dt> <dd>Definition 1</dd> <dt>Term 2</dt> <dd>Definition 2</dd> <dt>Term 3</dt> <dd>Definition 3</dd></pre>

		<pre><dt>Term 3</dt> <dd>Definition 3</dd> </dl></pre>
--	--	--

'Nested' lists

It is also possible to 'nest' a list within a list.

e.g. if an unordered list (ul) is placed inside one of the items of an ordered list (ol), as follows:

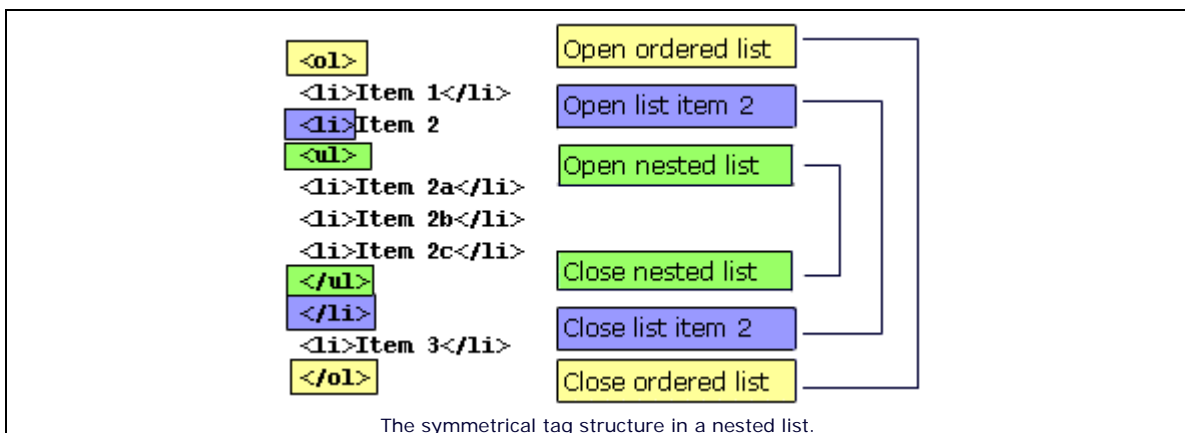
```
<ol>
<li>Item 1</li>
<li>Item 2
<ul>
<li>Item 2a</li>
<li>Item 2b</li>
<li>Item 2c</li>
</ul>
</li>
<li>Item 3</li>
</ol>
```

The result is a nested list as follows:

1. Item 1
2. Item 2
 - o Item 2a
 - o Item 2b
 - o Item 2c
3. Item 3

It is important to remember to close all tags in a list correctly and maintain symmetrical tagging. It can become difficult to track the tags in a nested list, and any unclosed or unsymmetrical tags may affect the way the list renders in a browser.

e.g. The tags for a basic nested list should be symmetrical, as follows:



Tables

Tables can be used in web pages for the presentation of data under headings. Because they allow for the precise placement of information on the screen, they are often also used to provide the overall layout for web pages. In either case, the basic elements, attribute and values used to create the table and its rows and cells are the same.

Table elements

The element tags for the creation of a table are as follows:

<code><table>...</table></code>	The beginning and end of a table.
<code><tr>...</tr></code>	The beginning and end of a row.
<code><td>...</td></code>	The beginning and end of a piece of table data (a cell).

Thus, the following code creates a 2x2-cell table:

```
<table>
<tr>
<td>cell 1</td>
<td>cell 2</td>
</tr>
<tr>
<td>cell 3</td>
<td>cell 4</td>
</tr>
</table>
```

cell 1	cell 2
cell 3	cell 4

Further columns can be added by increasing the number of `<td>...</td>` tags (New lines are marked with ►):

```
<table>
<tr>
<td>cell 1</td>
<td>cell 2</td>
►<td>cell 3</td>
</tr>
<tr>
<td>cell 4</td>
<td>cell 5</td>
►<td>cell 6</td>
</tr>
</table>
```

cell 1	cell 2	cell 3
cell 4	cell 5	cell 6

and further rows can be added by increasing the number of <tr>...</tr> tags (New lines are marked with ►):

```
<table>
<tr>
<td>cell 1</td>
<td>cell 2</td>
</tr>
<tr>
<td>cell 3</td>
<td>cell 4</td>
</tr>
► <tr>
► <td>cell 5</td>
► <td>cell 6</td>
► </tr>
</table>
```

cell 1	cell 2
cell 3	cell 4
cell 5	cell 6

Table layout

The browser renders the table width as the maximum number of cells on any one row. It then lines the cells on other rows up with these cells. If a row has fewer cells than this maximum, it will not automatically stretch these cells to fill the space but will line them up with the ones in the longest row, and leave an empty space in the table.

Thus a table with 2 cells on row 1, 3 cells on row 2, and 1 cell on row 3, as follows:

```
<table>
<tr>
<td>cell 1</td>
<td>cell 2</td>
</tr>
<tr>
<td>cell 3</td>
<td>cell 4</td>
<td>cell 5</td>
</tr>
<tr>
<td>cell 6</td>
</tr>
</table>
```

would produce the following (a background has been applied to the cells to show the spacing clearly):

cell 1	cell 2	
cell 3	cell 4	cell 5
cell 6		

In order to stretch the cells to fill the space in the table, the column-span attribute (colspan) is used.

Thus, if colspan = "2" is added to the HTML creating the first row as follows:

```
<tr>
<td>cell 1</td>
<td colspan="2">cell 2</td>
</tr>
```

the result is:

cell 1	cell 2	
cell 3	cell 4	cell 5
cell 6		

Similarly, if colspan="3" is added to the <td> tag in the third row, the result is:

cell 1	cell 2	
cell 3	cell 4	cell 5
cell 6		

In the same way, the row-span attribute (rowspan) is used to control the spacing of rows.

If rowspan = "2" is added to the HTML creating the second row as follows:

```
<tr>
<td>cell 3</td>
<td rowspan="2">cell 4</td>
<td rowspan="2">cell 5</td>
</tr>
```

the result is:

cell 1	cell 2	
cell 3	cell 4	cell 5
cell 6		

Note that this only works down a table. Adding **rowspan = "3"** to cell five will only be effective if the cell is moved to the first row as follows:

```
<table>
<tr>
<td>cell 1</td>
<td>cell 2</td>
<td rowspan="3">cell 5</td>
</tr>
<tr>
<td>cell 3</td>
```

```

<td rowspan="2">cell 4</td>
</tr>
<tr>
<td>cell 6</td>
</tr>
</table>

```

cell 1	cell 2	cell 5
cell 3	cell 4	
cell 6		

It is also possible to place a new table within a table cell.

```

<table>
<tr>
<td>cell 1
<table>
<tr>
<td>cell a</td>
<td>cell b</td>
</tr>
<tr>
<td>cell c</td>
<td>cell d</td>
</tr>
</table>
</td>
<td>cell 2</td>
</tr>
<tr>
<td>cell 3</td>
<td>cell 4</td>
</tr>
</table>

```

Cell 1	cell 2	
cell a		cell b
cell c		cell d
cell 3	cell 4	

A great deal of control can be added to the spacing of tables though the use of tables within table cells, alongside colspan and rowspan. It is thus common to use them to create the overall layout for webpages.

Where this is done, however, it is important to be aware of how the information may be presented for users of text-only browsers or screen-reading software, which 'read' the information from left to right.

For example, in the following table, a text-only browser would be likely to present the information from cell 5 before cells 3 and 4.

cell 1	cell 2	cell 5
cell 3	cell 4	
cell 6		

It is also important to use relative sizing (e.g. in percentages) rather than absolute sizing (e.g. in pixels) to ensure that the table will be visible on a range of screen sizes. (See the 'Key design issues' section of this technical guide for further information on this issue).

Attributes and values

The attributes used to format the style and layout of tables are applied to the table tag `<table attribute = "value">...</table>` and the table cell tags `<td attribute="value">...</td>`.

Table attributes and values

Attribute	Possible values	Comment
width="value"	A value in terms of the percentage of the screen width (relative width) e.g. width="90%". or in pixels (absolute width) e.g. width="550".	Care must be taken when using absolute width to ensure that the table can be viewed on smaller screens. (see the 'Key design issues' section of this technical guide).
border="value"	a value in pixels.	A zero border value is likely to be used where tables are used for page layout and positioning.
cellspacing="value"	a value in pixels.	Defines the space between table cells.
cellpadding="value"	A value in pixels.	Defines the space within table cells between the cell border and the content.
align="value"	left, center, right.	Used to align the entire table on the page. Only needed where a left-alignment is not required as this is the default alignment.
bgcolor="value"	A hash-mark (#), followed by a six-figure 'hexadecimal' colour code.	Sets the background colour of the table.
bordercolor="value"	A hash-mark (#), followed by a six-figure 'hexadecimal' colour code.	Sets the outside border colour of the table.

Table cell attributes and values

Attribute	Possible values	Comment
width="value"	A value in terms of the percentage of the table width (relative width) e.g. width="50%". or in pixels (absolute width) e.g. width="250".	relative sizing for table cells can be used to set a proportional width within absolutely or relatively-sized tables.
height="value"	a value in pixels.	Sets the height of individual cells (The entire row takes the height of the largest cell).
align="value"	left, center, right.	Sets the horizontal alignment of individual cells.
valign="value"	top, middle, bottom.	Sets the vertical alignment of individual cells.
bgcolor="value"	A hash-mark (#), followed by a six-figure 'hexadecimal' colour code.	Sets the background colour of the table.
bordercolor="value"	A hash-mark (#), followed by a six-figure 'hexadecimal' colour code.	Sets the border colour of individual cells.

Introduction to Cascading Style Sheets

Introduction

CSS (Cascading Style Sheets - also referred to simply as 'Style Sheets') provide a means of adding design elements to basic HTML pages. For example, using CSS, it is possible to control the colour, positioning and spacing of objects such as text, links, images and tables. All the main design elements of this website are produced through the use of Style Sheets.

The use of CSS separates the presentation of web content from the structure, bringing three major benefits:

1. **Design options are increased** because CSS can provide more precise and wide-ranging control over design elements on the page (though it is important to make allowances for the fact that different browsers may interpret Style Sheet information differently and to thoroughly test pages).
2. **Changes are easier to make during design and development** because Style Sheet information is applied across all elements of a page or site. For example, if you want to change the style of all the links, it is necessary only to change the Style Sheet, rather than changing every link on the page or site.
3. **Accessibility is increased** as users can choose how they wish the site to appear by applying their own Style Sheets to a page, or by accessing the site content only without the style information. (It is thus important to design pages in such a way that the content remains accessible when the design features are removed, and to test that this is the case - e.g. by ensuring that colour is not used to impart meaning that can not be accessed when the colour is removed).

This page will provide the knowledge required to produce a Style Sheet and apply it to an HTML page. It does not aim to provide comprehensive coverage of CSS, but to provide an introduction to the basics.

As with the production of HTML pages, the only requirement is that you have a simple text editor, such as notepad for windows, and a browser such as MS Explorer in which you can test your pages. However, most WYSIWYG (What You See Is What You Get) software packages such as Macromedia Dreamweaver or Microsoft FrontPage allow for the automation of some aspects of CSS creation.

Learning activity: CSS in action

The following extract from a webpage uses CSS for layout, font, and colours. The HTML document is reproduced below, with the different sections divided up and labeled.

My Page

Welcome to my web page

It is styled using CSS

Hope you like it

Examine the following HTML document and try to establish where the CSS information is located, what effect it has and how it links to the HTML. Try to notice the patterns in the syntax and punctuation of the CSS. When you have studied the document, refer to the explanations of the different sections of the document beneath.

You can also use this activity to check your knowledge of the structure of HTML documents and tags, by identifying the function of all the different tags, before checking the explanations.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>CSS in action</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-
8859-1">
<style type="text/css">
body {
color: #003;
background-color: #ff9;
}
h1 {
font-family: Verdana, Arial, Helvetica, sans-serif;
font-size: 120%;
font-weight: bold;
color: #006;
}
p {
font-family: Verdana, Arial, Helvetica, sans-serif;
font-size: 90%;
}
.red {
font-family: Verdana, Arial, Helvetica, sans-serif;
font-size: 90%;
color: #f00;
font-style: italic;
}
</style>
</head>
<body>
<h1>My Page</h1>
<p>Welcome to my web page</p>
<p class="red">It is styled using CSS</p>
<p style="text-align: center; color: #906;"> Hope you like it</p>
</body>
</html>
```

Explanations

Section of code	Explanation
<code><!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"></code>	The DOCTYPE (document type) definition, known as a DTD. This declares what type of page it is and what language is being used, and it allows the page to be validated as conforming to Worldwide Web Consortium (W3C) standards.
<code><html></code>	The 'open HTML' tag. Tells the browser that what follows is an HTML document.
<code><head></code>	The 'open head' tag. The head contains information which is basically not intended for display. It is loaded into the browser before the body section. CSS rules are placed between these tags.
<code><title>CSS in action</title></code>	The title. This states what the title of the page is, allows the title to be displayed at the top of the browser window, and provides a title that can be saved to a user's favourites list.
<code><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"></code>	Meta command. These commands provide extra information about the pages. This meta command describes the type of content and which set of characters is in use.
<code><style type="text/css"></code>	The 'open style' tag. The style rules must be placed within style tags or in a separate file linked to the document.
<code>body { color: #003; background-color: #ff9; }</code>	Style rule that applies the following style to all the tags in the body of the HTML page(s) linked to the Style Sheet: Dark blue text colour Light yellow background colour [NB. The full hexadecimal code for these colours are #000033 (dark blue) and #ffff99 (light yellow), but these shortened (#003 and #ff9) codes can be used in Style Sheets].
<code>h1 { font-family: Verdana, Arial, Helvetica, sans-serif; font-size: 120%; font-weight: bold; color: #006; }</code>	Style rule that applies the following style to all the <code><h1></h1></code> tags contained in the HTML page(s) linked to the Style Sheet: They are displayed in the first font in the font-family list that is available on the user's computer; The text is sized at 120% of the default size, in bold and in a blue colour.
<code>p { font-family: Verdana, Arial, Helvetica, sans-serif; font-size: 90%; }</code>	Style rule that applies the following style to all the <code><p></p></code> tags contained in the HTML page(s) linked to the Style Sheet: They are displayed in the first font in the font-family list that is available on the user's computer; The text is sized at 90% of the default size.
<code>.red { font-family: Verdana, Arial, Helvetica, sans-serif; font-size: 90%; color: #f00; font-style: italic; }</code>	A style 'class' that is not tied to a particular HTML element. This style rule can be applied to any tag contained in the HTML page(s) linked to the Style Sheet. The style is applied to the tag in an HTML document by labeling it with the name 'red', e.g. <code><h3 class="red">A red heading</h3></code>
<code></style></code>	The 'close style' tag. The style rules must be placed within style tags or in a separate file linked to the document.
<code></head></code>	The 'close head' tag. The head contains information which is basically not intended for display. It is loaded into the browser before the body section. CSS rules are placed between these tags.
<code><body></code>	The 'open body' tag. The body of an HTML document contains the main display content. It is here that text, images, links, form elements, tables and lists are placed.
<code><h1>My Page</h1></code>	An <code><h1></h1></code> heading. This takes the style set in the h1 style rule above.
<code><p>Welcome to my web page</p></code>	A paragraph. This takes the style set in the p style rule above.
<code><p class="red">It is styled using CSS</p></code>	A paragraph with the style specified in the style class '.red' above.
<code><p style="text-align:center; color: #906;"> Hope you like it</p></code>	A paragraph with the style specified 'inline'. This takes the style set in the p style rule above, and alters or adds the style set in the line. Thus the font family and size set in the p style rule above apply to this paragraph and further styles are also added (centred text and a purple colour). Where the styles are in conflict, the inline style overrules the style set in the style tags.
<code></body></code>	The 'close body' tag. The body of an HTML document contains the main display content. It is here that text, images, links, form elements, tables and lists are placed.
<code></html></code>	The 'close HTML' tag. Required to end an HTML document.

Linking CSS to HTML documents

As shown in the activity above, Style Sheets consist of a set of **rules** which provide information on how particular elements on a page should be displayed. The style information can be linked to the HTML document using the following three methods:

1. Embedded CSS

The style rules are placed in the document head between the following tags:

```
<style type="text/css">
```

and

```
</style>
```

2. External CSS

The rules are placed in a separate text file (without style tags) which is saved with a .css extension. The file can then be linked to the HTML document by placing the following in the head of the document:

```
<link href="filename.css" rel="stylesheet" type="text/css">
```

It is also possible to link to the CSS file by importing the file when the page loads, using the following syntax:

```
<style type="text/css"><!--  
@import url("mystyle.css");  
--></style>
```

Because older browsers do not recognise the @import syntax, it is common to use both methods together to link to different CSS files depending on what kind of browser is being used. If a Style Sheet designed for older browsers is placed in an href link, followed by an @import link, modern browsers will override the first Style Sheet with the second, while older browsers will use the first and ignore the second.

3. Inline CSS

Style information is added to an HTML tag in a similar way to that in which a range of attributes and values can be added (see the HTML section). e.g.

```
<p style="font-weight: bold; color: #009"; text-align: center;>  
Text styled with inline CSS</p>
```

produces:

Text styled with inline CSS

The term 'Cascading Style Sheets' derives from the fact that the Style Sheet information from all three methods can work together with the information from the latter overriding the information from the former in a 'cascade' (See 'the cascade' section below).

It is worth noting that the latest standards of XHTML recommend using external CSS only rather than using inline or embedded styles. This is because external Style Sheets allow the maximum separation of content from presentation, so that all content information is

effectively placed in one file and all presentation placed in another. This makes it easier for users to display only the content or to apply their own Style Sheet.

Style Sheet syntax

The **rules** that Style Sheets are made up of consist of the following elements:

Selectors - A references to which elements on the HTML page the style should be applied to, or to the name of a style 'class' which can be applied to any tag (see the 'creating CSS classes' section below).

Declarations - A series of statements about what the style should be. These are made up of **properties** and **values**.

The syntax is as follows:

```
selector {  
property 1: value(s);  
property 2: value(s);  
}
```

e.g.

```
p {  
font-family: Verdana, Arial, Helvetica, sans-serif;  
color: #006;  
}
```

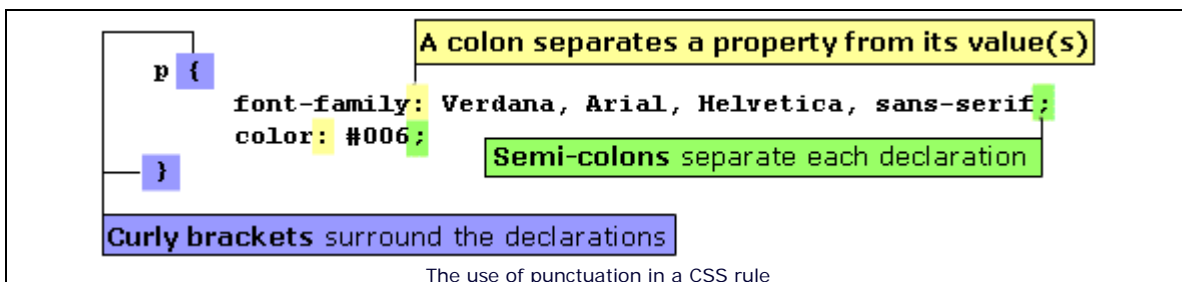
applies a style to all the <p></p> tags contained in the HTML page(s) linked to the Style Sheet. They are displayed in the first font in the font-family list that is available on the user's computer, and in a blue colour.

The same syntax can be added inline to a tag in an HTML document by replacing the curly brackets with 'style=' and quotation marks as follows:

```
<p style="font-family: Verdana, Arial, Helvetica, sans-serif;  
color: #006;">Add text here</p>
```

Punctuation and spacing

A CSS rule must be accurately punctuated to allow the browser to recognise and distinguish between different selectors, declarations, properties and values.



It is common to space Style Sheet rules as shown above with each declaration on a new line and indented from the selector. This is done for clarity to make reading and editing Style

Sheets easier, but it is not obligatory and has no effect on how the browser interprets the information.

Creating CSS Classes

Different styles can be added to HTML elements through the use of classes. This is done by adding an extension to the selectors in the Style Sheet, placing the extension name after the HTML element name and a full-stop. The styles can then be applied by referring to the extension in the HTML document.

e.g. Two different paragraph styles can be created as follows:

```
p.bluesans {  
font-family: Verdana, Arial, Helvetica, sans-serif;  
color: #006;  
}  
p.redserif {  
font-family: Georgia, Times New Roman, Times, serif;  
color: #f00;  
}
```

The styles can then be applied to different paragraphs in the HTML document, as follows:

```
<p class="bluesans">This is a paragraph with the "bluesans" style  
applied.</p>  
  
<p class="redserif">This is a paragraph with the "redserif"  
style</p>
```

which produces the following:

```
This is a paragraph with the "bluesans" style applied.  
This is a paragraph with the "redserif" style applied.
```

Classes can also be created without attachment to a particular HTML element. The procedure is the same as above, but instead of adding an extension to an HTML element, it is created independently. To do this, a full-stop is added before the class name as follows:

```
.bluesans {  
font-family: Verdana, Arial, Helvetica, sans-serif;  
color: #006;  
}  
.redserif {  
font-family: Georgia, Times New Roman, Times, serif;  
color: #f00;  
}
```

The styles can then be applied to different elements in the HTML document, as follows:

```
<h2 class="redserif">This is a header (<h1>) with the "redserif" style applied.</h2>

<p class="bluesans">This is a paragraph with the "bluesans" style applied.</p>

<h4 class="bluesans">This is a header (<h2>) with the "bluesans" style applied.</h4>

<p class="redserif">This is a paragraph with the "redserif" style applied.</p>
```

which produces the following:

This is a header (<h2>) with the "redserif" style applied.

This is a paragraph with the "bluesans" style applied.

This is a header (<h3>) with the "bluesans" style applied.

This is a paragraph with the "redserif" style applied.

The styles will be applied to the elements and will override any conflicting colour and font styles that are already applied to them. Any styles that do not conflict will also be maintained through the 'cascade' (see section below).

Classes with <div> and tags

<div></div> tags can be used in the HTML document to create a 'division' in the page in which all elements will have a particular style attached.

e.g.

Style Sheet:

```
.rightbold {
text-align: right;
font-weight: bold;
}

.centeritalic {
text-align: right;
font-style: italic;
}

.bluesans {
```

```
font-family: Verdana, Arial, Helvetica, sans-serif;
color: #006;
}
.redserif {
font-family: Georgia, Times New Roman, Times, serif;
color: #f00;
}
```

HTML Document:

```
▶ <div class="rightbold">
<h2 class="redserif">This is a header (<h1>) with the "redserif"
style applied.</h2>
<p class="bluesans">This is a paragraph with the "bluesans" style
applied.</p>
<p>in both cases, the "rightbold" style is applied.</p>
▶ </div>
▶ <div class="centeritalic">
<h4 class="bluesans">This is a header (<h2>) with the "bluesans"
style applied.</h4>
<p class="redserif">This is a paragraph with the "redserif" style
applied.</p>
<p>The "centeritalic" style is applied in both cases.</p>
▶ </div>
```

The styles linked to the `<div></div>` tags (shown next to the ▶ arrows) are applied to all the tags within, as follows:

This is a header (<h2>) with the "redserif" style applied.

This is a paragraph with the "bluesans" style applied.

In both cases, the "rightbold" style is applied.

This is a header (<h3>) with the "bluesans" style applied.

This is a paragraph with the "redserif" style applied.

The "centeritalic" style is applied in both cases.

`` tags are used in a very similar way and also apply style to a section of a document. However, while `<div>` tags are always followed by a line-break, `` tags are not.

`` tags can thus be used to apply different styles to text within sentences and paragraphs.

e.g.

```
<p>Using &lt;span&gt; tags it is possible to apply the <span
class="bluesans">"bluesans" style</span> and the <span
class="redserif">"redserif" style in the same line.</span></p>
```

produces:

Using `` tags it is possible to apply the "bluesans" style and the "redserif" style in the same line.

Replacing the `` tags with `<div>` tags produces the following:

Using `` tags it is possible to apply the

"bluesans" style
and the
"redserif" style
in the same line.

The cascade

Cascading Style Sheets are particularly useful in that they allow a set of generic styles to be applied to elements of an entire site. Changes can be made across the whole site simply by making one change to an external Style Sheet which all the pages are linked to.

e.g. If the following rule is included in an external Style Sheet linked to all the pages of a site

```
h2 {
font-family: Verdana, Arial, Helvetica, sans-serif;
font-size: 140%;
color: #006;
background-color: #ff9;
font-weight: bold;
text-align: center;
}
```

all the `<h2></h2>` headings will appear as follows:

Example header

However, where the designer wishes to add particular styles to individual pages or sections, or to make a generically-styled element on a particular page look different, s/he can take advantage of the 'cascade' which allows Style Sheet information from different sources to work together.

Where one change is required on a particular page, this can most easily be achieved by adding an inline style rule.

e.g. Adding the following syntax maintains the information from the generic Style Sheet, but replaces conflicting rules (concerning colour) with the inline style.

```
<h2 style="color: #303;">Example header</h2>
```

Thus a purple text colour is applied as follows:

Example header

Where a change is needed a number of times, an embedded style rule can be added, or a second external Style Sheet can be included.

Thus, adding the following in the head of the document maintains the external style information with the exception of the conflicting style rules (background colour and alignment):

```
<style type="text/css">
h2 {
background-color: #ff0;
text-align: left;
}
</style>
```

and all <h2></h2> headers on the page will appear with a left alignment and a bright yellow background, as follows:

Example header

The effect would be the same if this information were added to a second external Style Sheet with a link placed after the first in the head of the document. The information in the second will work together with that in the first, overriding any rules that refer to the same properties, but preserving any other rules.

The browser will apply any inline styles, before applying embedded styles and finally applying external styles. Where there is style information from different sources that conflicts with each other, this is the order of precedence. Inline styles will overrule embedded styles which will overrule external styles. Styles from any source that do not conflict will be preserved.

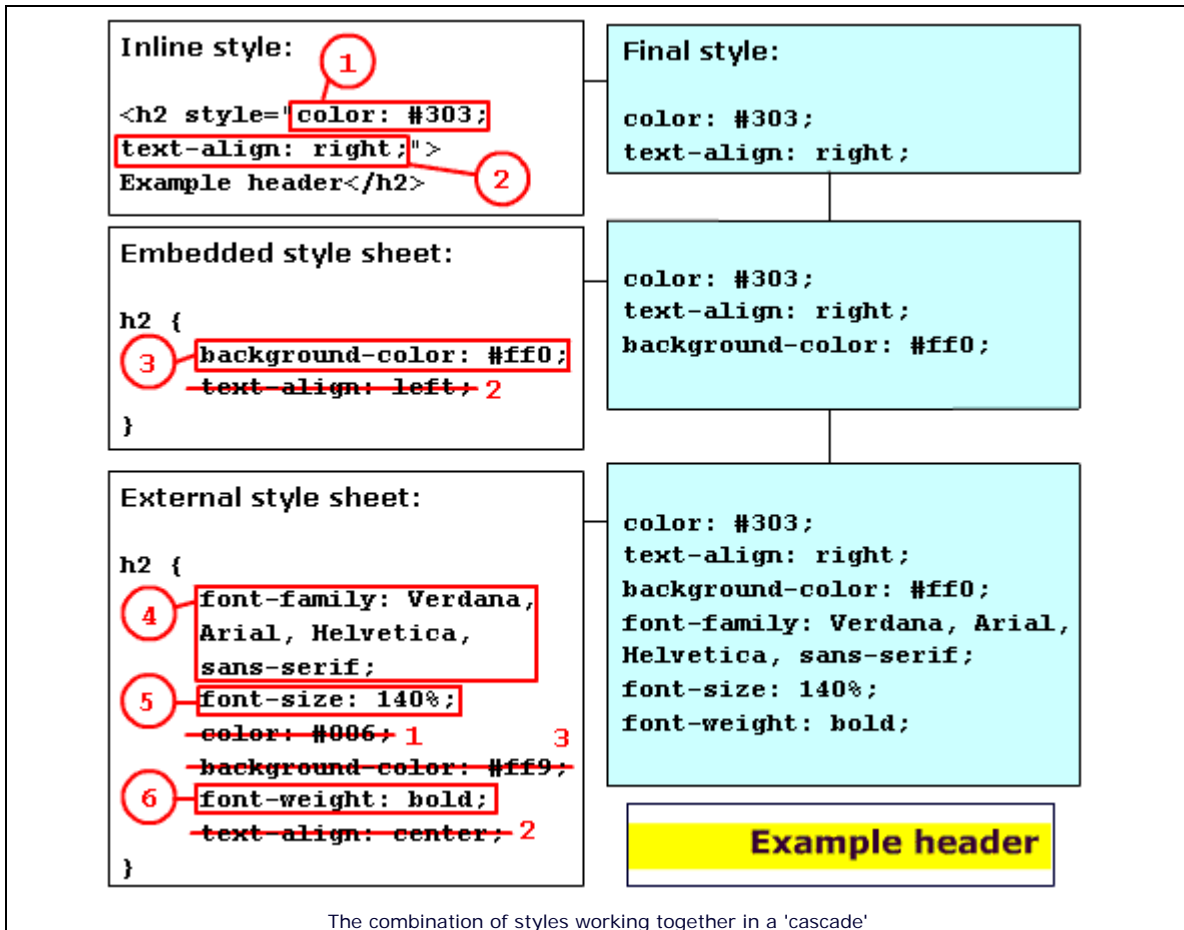
Thus if the following inline style is added to a heading on a page with the style information above :

```
<h2 style="color: #303; text-align: right;">Example header</h2>
```

the following heading results:

Example header

The information from the three sources 'cascades' as follows:



Inheritance

HTML documents can be thought of as having a family tree structure where different elements are the parent or child of other elements. Thus for example, the **<body>** element is the parent of all other tags, and the list item tags (****) are the children of the list tags (**** or ****).

Most of the styles that are applied to the parent element will be inherited by the child element. This means that if a particular rule has been applied to the parent, it is not necessary to apply it again to the child element.

e.g.

In the following Style Sheet, the font-family information can be placed in the body selector, removing the need to repeat it in the other tags. The inheritance of the colour and font-family information is overridden in the 'h1' and '.red' selectors by specifying an alternative colour and family.

```
<style type="text/css">
body {
color: #003;
background-color: #ff9;
}
p {
font-family: Verdana, Arial, Helvetica, sans-serif;
font-size: 90%;
}
h1 {
font-family: Verdana, Arial, Helvetica, sans-serif;
font-size: 120%;
color: #006;
font-weight: bold;
}
.red {
font-family: Georgia, Times New Roman, Times, serif;
font-size: 90%;
color: #f00;
}
```

Thus, the following Style Sheet produces exactly the same results.

```
<style type="text/css">
body {
color: #003;
background-color: #ff9;
font-family: Verdana, Arial, Helvetica, sans-serif;
}
p {
font-size: 90%;
}
h1 {
font-size: 120%;
color: #006;
font-weight: bold;
}
```

```
}  
  
.red {  
font-family: Georgia, Times New Roman, Times, serif;  
font-size: 90%;  
color: #f00;  
}
```

Careful use of inheritance can provide a means of making CSS information smaller and more efficient.

Resources for further development

Once you have an understanding CSS, one of the most important resources for use when working with Style Sheets is a reference to the properties and possible values that can be applied to different elements of an HTML page. A clear example is provided at http://www.w3schools.com/css/css_reference.asp, which offers further information on the use of different properties.

W3schools also provides tutorials, examples and quizzes at <http://www.w3schools.com/css/default.asp>

The Worldwide Web Consortium (W3C)'s CSS page at <http://www.w3.org/Style/CSS/> offers a wealth of information, and provides an opportunity to validate your CSS at <http://jigsaw.w3.org/css-validator/>. This makes it possible to check that your CSS meets web standards and guidelines by entering a link to your CSS file, uploading your file from your computer, or pasting your CSS into a text box on the page.

A useful article by John Gallant and Holly Bergevin on using CSS 'short hand' properties to reduce the size of CSS files and increase efficiency is also available at <http://www.communitymx.com/content/article.cfm?cid=90F55>

Web forms

Introduction

Web forms are at the heart of online questionnaires which basically consist of text (with any associated multimedia) and form elements inserted into a web page. Forms make it possible for participants to enter information and send this information to the researcher for analysis.

This page will provide the knowledge required to produce a web form. It will introduce the different form elements available and cover the different options for how these elements function and appear.

Learning activity: web forms

Examine the action of the different form elements in the following form. Then complete the tasks that follow.

1. What is your name?

2. How old are you?

 years

3. How often do you use the internet?

- everyday
- 2-3 days per week
- 4-5 days per week
- 6-7 days per week
- less than once a week

4. Which of the following do you regularly use the internet for?

(You can select as many options as you like. If you would like to remove a selection you have made, select it again to deselect it).

- E mail
- Finding information about things to buy
- Making purchases
- Entertainment
- Finding general information
- Educational courses
- Downloading music

- Discussion boards
- Real-time chat

5. How would you rate your skill as an internet user?

6. What, in your opinion, are the three main advantages of the internet?

Task 1: The basics

A. Check your knowledge of the names of different form elements by matching the names (1-5) to the examples (a-e) below. Then select 'Answers' to check.

Form elements

1. Text box
2. Textarea
3. Select box
4. Check boxes
5. Radio buttons

Examples

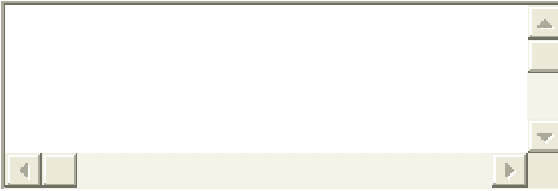
a.

b.

- E mail
- Finding information about things to buy
- Making purchases

c.

d.



e.



everyday



2-3 days per week



4-5 days per week

Answers

1=c

2=d

3=a

4=b

5=e

B. Consider the following questions about the web form, before reading the answers.

1. What is the difference between the action of radio buttons and check boxes? Which have the same actions as the select box?
2. Are there any maximums to the length of the text that can be entered into the text boxes and textarea? What happens when the text reaches the end of the boxes?

Answers

1. The radio buttons work together as a group. It is only possible to activate one option at a time. However, multiple Check box options can be chosen simultaneously.

The radio buttons are closest to the select box in that only one option can be chosen. It is, however, possible to set a select box so that multiple selections can be chosen by holding down the control button (see section below).

2. The second text box has been set with a maximum character length of 3 characters. If no maximum character length is set (as is the case with the first text box), text can be entered beyond the width of the box. Maximum character lengths can also be set for textareas. If no maximum is set, the scroll bar is automatically activated once the end of the box is reached.

Task 2: Examining the HTML

Examine the HTML underlying the web form and consider how it is rendered in the browser. Try to establish what the functions of the different parts of the HTML are. When you have studied the document, refer to the explanations of the different sections of the document beneath.

You can also use this activity to check your knowledge of the structure of HTML documents and tags, by identifying the function of all the different tags, before checking the explanations.

```
<form name="form1" action="" method="post">
<p> 1. What is your name?</p>
<p><input type="text" name="namebox" /></p>
<p>2. How old are you?</p>
<p><input name="agebox" type="text" size="5" maxlength="3" />
years</p>
<p>3. How often do you use the internet? </p>
<p><input type="radio" value="Everyday" name="often" />
Everyday<br />
<input type="radio" value="2-3 days" name="often" />
2-3 days per week<br />
<input type="radio" value="4-5 days" name="often" />
4-5 days per week</p>
<p>4. Which of the following do you regularly use the internet
for?<br />
(You can select as many options as you like. If you would like to
remove a selection you have made, click on it again to deselect
it).</p>
<p><input type="Check box" value="Email" name="email" />E mail<br
/>
<input type="Check box" value="Finding info" name="FI" />Finding
information about things to buy<br />
<input type="Check box" value="Purchases" name="purchases"
/>Making purchases</p>
<p>5. How would you rate your skill as an internet user?</p>
<p><select name="levelSelect">
<option selected="selected">Select an option</option>
<option>-----</option>
<option value="VA">Very advanced</option>
<option value="A">Advanced</option>
</select></p>
<p>6. What, in your opinion, are the three main advantages of the
internet?</p>
<textarea name="advantages" rows="5" cols="50"></textarea>
<p><input type="button" name="Submit" value="Submit"
onclick="openResponse();"></p>
</form>
```


Explanations

Section of code	Explanation
<code><form name="form1" action="" method="post"></code>	Tag to open the form, which is given a unique name to enable any scripts to refer to the form(s) in a page, and the elements within. The action and method attributes control how and where the information is sent when the form is submitted (see section below).
<code><p><input type="text" name="namebox" /></p></code>	Text for question 1 within paragraph tags. See 'Ensuring web forms are accessible' below for information on using labels for form elements
<code><p><input type="text" name="namebox" /></p></code>	Input tag which places a text box (type="text") on the page of the default size and without a maximum number of characters. The name allows the box and its contents to be accessed by any scripts to validate or process the form.
<code><p>2. How old are you?</p></code>	Text for question 2 within paragraph tags.
<code><p><input name="agebox" type="text" size="5" maxlength="3" /> years</p></code>	Input tag for a text box (type="text") with a 5-character width and the maximum number of characters set to 3. The close tag (' />') ensures that the input tag meets the latest standards of XHTML which state that all tags should be closed.
<code><p>3. How often do you use the internet?</p></code>	Text for question 3 within paragraph tags.
<code><p><input type="radio" value="Everyday" name="often" /> Everyday
<input type="radio" value="2-3 days" name="often" /> 2-3 days per week
<input type="radio" value="4-5 days" name="often" /> 4-5 days per week</p></code>	Input tags for three radio buttons (type="radio"). The fact that they are given the same name (name="often") means that they operate as a group whereby only one option can be selected at any one time. This name means that scripts can access the tags to check which of the buttons with the same name has been selected. The value of the selected button can then be extracted.
<code><p>4. Which of the following do you regularly use the internet for?
(You can select as many options as you like. If you would like to remove a selection you have made, click on it again to deselect it).</p></code>	Text for question 4 within paragraph tags.
<code><p><input type="Check box" value="Email" name="email" />E mail
<input type="Check box" value="Finding info" name="FI" />Finding information about things to buy
<input type="Check box" value="Purchases" name="purchases" />Making purchases</p></code>	Input tags for three Check boxes (type="Check box"). The names act as a reference to each Check box and mean that scripts can access them to check whether they have been checked. The values of any checked boxes can then be extracted.
<code><p>5. How would you rate your skill as an internet user?</p></code>	Text for question 5 within paragraph tags.
<code><p><select name="levelSelect"><option selected="selected">Select an option</option><option>-----</option><option value="VA">Very advanced</option><option value="A">Advanced</option></select></p></code>	Select tags which create a select box with a unique name to allow it to be referenced. Each choice is placed within <option></option> tags which are given a value so that the value of the selected option can be extracted. The first two options are not intended to be selected - the first acts as an instruction and the second separates this from the genuine options. Inserting 'selected' after the first option means that this option is selected by default and appears in the select box when the form is loaded.
<code><p>6. What, in your opinion, are the three main advantages of the internet?</p></code>	Text for question 6 within paragraph tags.
<code><textarea name="advantages" rows="5" cols="50"></textarea></code>	Textarea tags which create a text input area, given a unique name with the 'name' attribute. The 'rows' attribute specifies the number of lines in the box, and 'cols' specifies its width.
<code><p><input type="button" name="Submit" value="Submit" onclick="openResponse();"></p></code>	Input tag for a button (type="button"), which has no default action and which must be associated with a script to have an effect. It is used with this form because it is processed using a piece of JavaScript code connected to the 'onclick' attribute (activated when the user clicks on the

	button). The value attribute sets the text on the button. For most forms, a submit button can be used (type="submit") which inserts a button with a default action of processing the form by sending the information to the place and in the manner specified by the form tag.
<code></form></code>	The 'close form' tag which ends the form.

The form tags

Form tags mark off the beginning and end of a form. Controls within the form tags are effectively grouped together so that when a submit button is clicked the data in all the controls within the form is sent for processing. It is possible to include multiple forms on a page, but only one form can be submitted at any one time.

A typical form tag is as follows:

```
<form name="form1" action="sendForm.php" method="post">
ADD FORM CONTROLS HERE...
</form>
```

The 'name' attribute allows the form to be given a unique name. This is essential in allowing the information from the form to be referenced by scripts so that the contents can be accessed and manipulated.

The 'action' attribute specifies where the information should be posted to when the 'submit' button is pressed. In the example above, the contents are posted to a page called sendForm.php, where it will be processed by a php script (for example, by being sent via email or sorted and transferred to a database).

The 'method' attribute specifies how the information in the form should be sent. The two values are 'post' or 'get'. The difference is that in the case of the 'post' method, the information is sent separately, while the 'get' method attaches the information to the end of the URL of the page that will process the information. An example of the 'get' method from a search engine in which the user has searched for the keywords 'JavaScript' and 'validation' and selected options to see pages only in English and from the UK, might be:

<http://www.searchengine.com?keywords=JavaScript+validation&lang=En&src=UK>

where '?' separates the URL from the data, '=' separates variable names from the data, and '&' separates different items of data.

Though the 'get' method can be useful in allowing the form data to be bookmarked along with the URL (allowing a search to be bookmarked), in the majority of cases, the 'post' option will be used as this can deal with larger amounts of data and does not reveal the data in the location bar of the browser.

Form controls for input

The majority of form controls that allow the user to input information are created using the input tag, with the type attribute specifying what type of control appears on the screen. Like an image tag, the input tag does not have a 'close' tag, and in order for it to meet the latest standards of XHTML, the close tag '/' must be included at the end. An example of an input tag for a text box is:


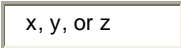

```
<input type="text" name="text box1" />
```


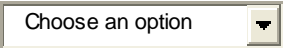
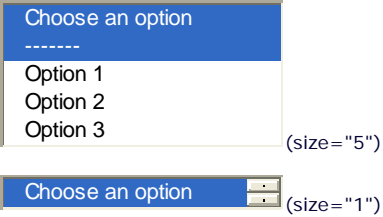
The form controls that are not created using the input tag are textarea and select box controls. These are displayed through `<textarea></textarea>` and `<select></select>` tags respectively.

Each control can be given attributes and values to control aspects such as width and length, and whether or not the control is filled in or selected by default.

As with any other HTML element, the appearance of forms and form controls can also be customised using Cascading Style Sheets (see the 'Introduction to CSS' section of this guide for more information).

The following table provides an example of how to insert the different types of control into a web page and outlines the different attributes and values that can be applied to each:

Name / Example	Example HTML	Comments / Possible attributes and values
Text box 	<pre><input type="text" name="text box1" size="15" maxlength="20" /></pre>	<p>This produces a single-line text input box with a width of 15 characters and the name 'text box1', that can hold a maximum of 20 characters.</p> <p>It is possible for text to appear in the box by default by adding value="x, y or z" to the tag, e.g.</p> 
Password box 	<pre><input name="password1" type="password" size="15" maxlength="10" /></pre>	<p>This produces a single-line text input box in which any characters input will display as asterisks or discs. (NB. This is a display feature which does not add to the security of information submitted by the form)</p> <p>Possible attributes and values are the same as for text boxes.</p>
Check box <input type="checkbox"/> Yes <input type="checkbox"/> No <input checked="" type="checkbox"/> Maybe	<pre><input type="Check box" name="c1" value="Yes"> Yes
 <input type="Check box" name="c2" value="No"> No
 <input type="Check box" name="c3" value="Maybe" checked="checked" /> Maybe</pre>	<p>This produces square tags that display a tick when selected and can allow multiple responses.</p> <p>When the form is submitted the name and value of any checked boxes will be submitted.</p> <p>Adding the attribute checked="checked" means that the box is selected by default.</p>
Radio button <input checked="" type="radio"/> Yes <input type="radio"/> No <input type="radio"/> Maybe	<pre><input type="radio" name="r1" value="Yes" / > Yes
 <input type="radio" name="r1" value="No" /> No
 <input name="r1" type="radio" value="Maybe" checked="checked" /></pre>	<p>Circular tags that fill in when one option is selected.</p> <p>Where a group of radio buttons are given the same name only one of the buttons can be selected at any one time.</p> <p>As with Check boxes, adding the attribute checked to one of the buttons means that it is selected by default when the page loads.</p>
Text area	<pre><textarea name="textareal"</pre>	<p>Works in the same way as the text box, but produces a multi-</p>

	<pre>cols="10" rows="5">Text can be added between the tags</textarea></pre>	<p>line text input box with a width and height specified by the cols and rows attributes.</p> <p>The box can be empty by default or text can be added between the tags</p>
<p>Select boxes</p> <p>Drop-down menu box</p> 	<pre><select name="select1"> <option selected="selected">Choose an option</option> <option>----- </option> <option>Option 1</option> <option>Option 2</option> <option>Option 3</option> </select></pre>	<p>An element which allows users to select options by clicking. One option is displayed at a time, and only one can be selected.</p> <p>In a similar way to Check boxes and radio buttons, adding the attribute selected="selected" to one of the options means that it is selected by default. It is a good idea to make the selected option an instruction to avoid measurement error if an option selected by default is submitted.</p>
<p>Select boxes</p> <p>List box</p> 	<pre><select name="select2" size="5" multiple="multiple"> <option selected="selected">Choose an option</option> <option selected="selected">--- ----</option> <option>Option 1</option> <option>Option 2</option> <option>Option 3</option> </select></pre>	<p>List boxes are produced using the same tags as drop-down menu boxes, but the attribute size="x" is added where x is the number of options visible at any one time.</p> <p>Adding the attribute multiple=multiple to the tag allows more than one option to be selected if the user holds down the control key whilst making selections (instruction on how to do this may be required or it may be preferable to use Check boxes which have a similar function).</p> <p>Adding the attribute selected=selected to one or more of the options means that it is selected by default.</p>

Hidden form fields

Hidden form elements are form controls that are not displayed on the page (though they are visible in the HTML source for the page). They are useful for storing and passing information from page to page which is not necessary or desirable to display. They can be thought of as text boxes with content that can be set by the developer via HTML or JavaScript rather than being completed by the user. The basic syntax for creating hidden form fields is as follows:

```
<input type="hidden" name="hidden1" value="value set via HTML or
JavaScript" />
```

The name attribute allows the field to be referenced by scripts so that the information (set using the value attribute) can be written to or collected from the field when needed.

Examples of uses of the hidden form control include the following:

Collecting information from questionnaires that span multiple pages so that the information from earlier pages can be passed to later pages without it being repeatedly

displayed on every page. Upon submission of the questionnaire on the final page, the data from a form on this page can be combined with one or more hidden elements containing the data from forms on previous pages. The data from multiple pages and forms can thus be submitted simultaneously. (There may be cases where it is desirable to submit information from each page separately to, for example, identify any key drop-out points).

Passing extra information about the submission alongside the data from the questionnaire. This may include information such as date and time of submission, approximate time taken to complete the questionnaire or individual questions, the page which contained the link the participant followed to reach the questionnaire, and information about the computer used by the participant such as the IP address or the browser used. These can provide possible routes to identifying and removing anomalous submissions (e.g. extremely rapid completion or multiple submissions) alongside providing information about the success of the forms of advertising used to elicit participation.

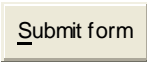
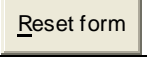
Passing information collected solely to enhance the effectiveness of data collection, management or processing. Examples may include the addition of a standard 'from' and 'subject' line to be added to emails containing data from particular questionnaires to facilitate automatic management of email, or, where appropriate, the inclusion of information about the version of a questionnaire completed.

See the '[Gathering information about participants](#)' section of this guide for further information about the use of hidden fields.

Buttons

Buttons must be added as the final component of a working web form. It is through selecting these buttons that the participant is able to submit or reset a form, or perform some other action programmed by the developer.

The three standard types of button are illustrated in the following table:

Name / Example	Example HTML	Comments / Possible attributes and values
Submit button 	<pre><input type="submit" name="Submit1" value="Submit form" /></pre>	<p>The submit button automatically processes the information in a form by sending it to the place and in the manner specified by the form tag.</p> <p>The text on a button can be changed by altering the value attribute.</p>
Reset button 	<pre><input type="reset" name="reset1" value="Reset form" /></pre>	<p>The reset button automatically returns all the form elements within the same form as the button to their default state as per when the document was opened. Use of a reset button must be carefully considered with longer forms in particular, as participants who accidentally press the reset button</p> <p>As before the text on the button can be changed by altering the value attribute.</p>
Standard button	<pre><input type="button" name="button1" value="Perform an action" onclick="perfActn();" /></pre>	<p>This type of button has no default action and it must be associated with a script to have an effect. The script will usually be linked to an 'onclick' event which will carry out the action set by the script when the button is pressed.</p> <p>The example shown would cause a JavaScript function called 'perfActn' to be activated when the button is clicked (this may, for example, be written to check that all required fields are completed before sending the information to the server (see the 'Form validation' section of this guide for further information).</p>

Using tables to organise controls into grids

The following example shows how groups of radio buttons can be organised into grids for Likert scales or semantic differential questions:

Complete the following statement by choosing the number that most closely matches your opinion for each row:

The internet is:

	1	2	3	4	5	
boring	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	interesting
difficult	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	easy
risky	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	safe
useless	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	useful

The radio buttons in each row are grouped by giving them the same name, so that only one option can be selected for each row. The buttons in each group are then given a different value so that when the form is submitted, the names are sent along with the value of the selected button (or no value if none are selected). This works in exactly the same way as four unrelated groups of radio buttons in four separate questions.

The table is then styled using HTML or CSS (See 'Introduction to HTML' / 'Introduction to CSS' sections of this technical guide).

Ensuring web forms are accessible

It is important to ensure that a questionnaire is fully accessible to users of non-graphical browsers or users who are not using a mouse or other click-and-point device. To do this, all form elements should be clearly labelled.

This is done using the `<label></label>` tags as follows:

```
<p><label>1. What is your name?</label></p>
<p><input type="text" name="namebox" /></p>
```

It is also important to make an explicit connection between the label and the control using the `for` attribute. This tells the user exactly which control the label refers to as follows:

```
<p><label for="name">1. What is your name?</label></p>
<p><input type="text" id="name" name="namebox" /></p>
```

The text box is given an identifying attribute `id="name"` which the label's `for` attribute explicitly refers to (`label for="name"`). This ensures that there will be no confusion in the use of the form for those using non-graphical browsers.

A second attribute which can be used to improve the accessibility of a questionnaire for those using a keyboard or equivalent is the **tabindex**. The tab key (or equivalent in, for example, voice activated browsers) can be used to select each element in a form in turn. The **tabindex** explicitly determines the order in which the elements are selected when the user tabs through them, as follows:

```
<input tabindex="2" type="text" name="box1">  
<input tabindex="1" type="text" name="box2">  
<input tabindex="3" type="submit" name="submit">
```

In this example, the second text box is selected first when the user presses the tab key. Pressing the key again selects the first box, and pressing it for the third time selects the submit button.

Introduction to JavaScript

Introduction

The use of JavaScript (or other scripting languages such as VB Script) can add a number of enhancements to an online questionnaire. These include:

1. Checking whether or not all questions have been answered correctly and prompting participants accordingly.
2. Playing a part in effective questionnaire design, for example, by allowing instructions to be delivered when needed in pop-up windows or alert boxes and allowing the use of browser detection to test which browser a user has and whether or not he or she has certain technology installed.
3. Improving the delivery of questions, for example by allowing data to be passed from page to page rather than being delivered on one long document, or by allowing randomisation.
4. Allowing the collection of data such as date and time of submission, and information about the computer used by the participant such as the IP address or the browser used.

The pages which follow this one provide an outline of some of these uses of JavaScript and offer a number of scripts which can be used or adapted. Beyond this website, there are also a number of sites offering freely-available scripts for direct use or adaptation. In order to take advantage of these resources, an understanding of the basics of JavaScript is generally required.

This section will introduce you to the basic concepts by examining two simple HTML documents with JavaScript functionalities. JavaScript will be the scripting language used throughout, though the procedures and scripts will be similar for other languages.

Accessibility

It is important to bear in mind that JavaScript may reduce the accessibility of the questionnaire. Users of text-only or screen-reading browsers are likely to be unable to access any functionality provided by JavaScript, and it is also possible that JavaScript may not be fully supported in the user's browser or that he or she may have chosen to deactivate it.

Thus, while it can be used to enhance the functionality of the questionnaire, it is not good practice to produce a questionnaire that is not usable without JavaScript. Where any key

functions are reliant on JavaScript (e.g. where the navigation and question delivery is controlled by JavaScript) it is important to inform the user of this and to offer alternatives.

The `<noscript></noscript>` tags can be used to deliver a message to users who are using a browser that does not support JavaScript. Anything placed between the tags will be displayed in cases where JavaScript is not available.

e.g.

```
<noscript>This questionnaire requires the use of JavaScript to function properly and it is not available in your browser. If you have disabled JavaScript in your browser settings, please enable it and refresh this page. If your browser does not support JavaScript and you would still like to complete the questionnaire, please email me for an alternative version.</noscript>
```

By placing the following meta tag in the head of the document between `<script></noscript>` tags, it is also possible to redirect the user to a new document. In this case the redirect occurs immediately (0 seconds) and `altpage.htm` is the name of the document the user is redirected to (which is held in the same folder as the original page). In this way, for example, users without JavaScript can be taken to a version of the questionnaire that does not require it, or to further instructions and contact details.

```
<meta http-equiv="refresh" content="0;url=/altpage.htm">
```

For accessibility, it is also important that JavaScript actions are not dependent on mouse actions such as dragging or double clicking. Where this the case, keyboard-accessible alternatives should also be provided. See 'Event handlers' section below.

Example 1: A welcome message

In the following example, the contents of a text box are inserted into an alert box when the user clicks on the button.

What is your name?

If this were the entire page, the HTML and JavaScript would be as follows:

```
<html>
<head>
<title>Hello</title>
<link href="mainstyle.css" rel="stylesheet" type="text/css">
<script language="javascript" type="text/javascript">
function showAlert() {
alert("Hello " + document.egForm1.namebox.value + ".");
}
</script>
</head>
<body>
<div class="ques">
<form name="egForm1" action="" method="post">
<p>What is your name?</p>
```



```
<p><input type="text" name="namebox" />
<input type="button" name="submit1" value="Submit"
onclick="showAlert();">
<input type="reset" name="reset1" value="Reset" /></p>
</form></div>
</body>
</html>
```

The majority of the document is HTML with some JavaScript added. Before considering the JavaScript, check your understanding of the HTML sections and, if necessary, read the brief explanation that follows.

Explanation

The HTML document has a head section that includes a title and a link to a Cascading Style Sheet called "**mainstyle.css**". The body contains a form which is styled using **<div class></div>** tags. These apply a style called "**ques**" which is defined in the Style Sheet to everything between the tags. The form contains three input elements - a text box called "namebox", a button called "submit" with "Submit" as the text (the value), and a reset button called "reset" with "Reset" as the value.

JavaScript within the HTML document structure

The main script is placed in the head of the document, between script tags as follows:

```
<script language="javascript" type="text/javascript">
```

and

```
</script>
```

It could also have been saved in a separate text file with the extension '.js'. This would be linked to the web page through a link placed in the head:

```
<script language="javascript" src="filename.js"
type="text/javascript"></script>
```

Functions

The action of the script takes place within a function called **showAlert** which is surrounded by curly brackets (**{}**).

```
function showAlert() {
alert("Hello " + document.egForm1.namebox.value + ".");
}
```

In this case, an alert box is shown with the contents of the text box incorporated into a message. The function can be explained as follows:

The value of the box is extracted and placed between two strings (text variables), namely "**Hello "** and **"."** (The quotation marks identify them as strings). The **alert** function is then triggered which places everything within the brackets in an alert box. The semi-colon (**;**) is placed at the end of each line of the function.

As in this example, a function is a block of code that carries out a particular action. In effect the code is not carried out until the function is 'called' from within the document when a particular event occurs (such as the user clicking a submit button).

The basic structure of a function is as follows:

```
functionName(){
Add function here...
}
```

The brackets following the function name allow different arguments to be included when it is called by different elements.

In the case of the example, this is not done - the code will always display the same message and the contents of the same text box when the function is called. However, the inclusion of arguments allows functions to be created which can be reused at different times by different elements in the document. In this case, a message and a reference to a particular text box name could be added when the function is called to allow it to display different messages along with the contents of different boxes.

They can thus be a very efficient way of carrying out different actions with one block of code (see 'Using arguments within functions' below).

This function is 'called' by the **onclick** code placed in the button tag:

```
<input type="button" name="submit" value="Submit"
onclick="showAlert();">
```

The **onclick** code is an example of an 'event handler' which triggers an action when a particular event occurs. In this case, the **showAlert()** function happens when the participant clicks on the button. Other examples of important event handlers are:

Event handler	Description
onDbIClick	Placed in the tag of a button or link, the action happens when the user uses the mouse to double select the object.
onmouseover / onmouseout	The action happens when the user places the mouse over a button or link / removes the mouse from the button or link.
onFocus / onblur	Usually placed in the tag of a form control, the action happens when the element receives focus (the user selects it) / loses focus (the user moves away).
OnChange	The action happens when a control loses focus (the user clicks away) and the value has been changed.
OnSubmit/onReset	Placed in the form tag, the action happens when a form is submitted (the user presses the 'submit' button) / reset (the user presses the reset button).
OnLoad / onunload	Placed in the body tag of a document, the action occurs when the page loads / unloads.

To ensure the accessibility of a questionnaire, any event handler that is dependent on the use of a mouse should be avoided. This will allow the JavaScript function to be activated by those using a keyboard or equivalent such as a voice activated browser. Where event handlers such as **onmouseover** or **onmouseout** are used, they should also be combined with **onfocus** or **onblur** events which will trigger the action when a user tabs to a button or similar element. For example in the following 'image flip' code, a function to change an image is called by the **onmouseover** event handler. The **onmouseout** handler calls a second function to change the image back. Adding the **onfocus="this.onmouseover();"** and the

`onblur="this.onmouseout();"` code ensures that the same actions will take place when a user without a mouse tabs to or away from the image.

```

```

Accessing data in controls

Elements in a web page are referenced by JavaScript using the 'tree-structure' of the document to refer to a particular element by name.

The contents of the text box in the example are referenced using the following:

`document.egForm1.namebox.value`

This refers to the **value** (the text) of the text box called **namebox** within the form called **egForm1** within the **document**.

By giving all the form controls on a page a unique name, it is possible to use this type of referencing to extract or set their value.

As an alternative to writing out the full reference where there are numerous references to controls within a form, it is also possible to make use of the **'this'** which is used in JavaScript to allow an object to refer to itself.

If an event handler is in the form tag, **'this'** can therefore replace the words **'document.egForm1'** in the reference.

If it is in a button within the form, the syntax **'this.form'** can be used to refer to the form (i.e. the form that contains this).

To refer to a text box control called **'box1'** from the button within the form, **'this.form.box1'** can be used instead of **'document.egForm1.box1'**. (i.e. box1 which is in the form that contains this).

This referencing is illustrated by the following example which is a form called **frm1** containing two text boxes called **box1** and **box2**, and a button called **btn1**. Code is added to the onclick event handler of the button which will display the contents of box1 in box2 when the user presses the button, as follows:

Box 1 Box2

The code is:

```
<p> Box 1 <input type="text" name="box1" />
<input type="button" name="btn1" value="Go"
onclick="document.frm1.box2.value=document.frm1.box1.value;" />
Box2 <input type="text" name="box2" /></p>
```

There is no difference if the full reference is replaced with the shortened version using **'this'**, as follows:

```
onclick="this.form.box2.value = this.form.box1.value";>
```

Although it is not a great deal shorter, using this syntax can make code more flexible and can avoid errors where the form name is referenced incorrectly (the **this.form** syntax

automatically feeds in the correct name). If it is used with arguments, it can make it easier to reuse functions with different forms and controls. (see section below)

Using arguments with functions

The use of arguments can be illustrated by the alert function which is used to display the name from the text box in example 1.

It is possible to create one function with an argument (given the name 'msg') that will be the message displayed when the function is called, as follows:

```
function showMessage(msg) {  
    alert(msg + ".");  
}
```

A string can be added to an **onclick** event handler which is then fed into the function when the function is called. This means that a different message can be displayed each time and it is not necessary to write a new function for each message.

e.g. All the buttons and links in the following paragraph use this same function to display a different message:

The following link is to **message 2**. It is an example of how a link can trigger an event in the same way as a button, using an 'empty' link to the JavaScript function as follows: ``

The code in the HTML document which calls the function is as follows:

```
<p><input type="button" name="btn2" value="Message 1"  
onclick="showMessage('This is message 1');" /><br />  
The following link is to <a href="JavaScript:void(0);" >  
onclick="showMessage('This is message 2');" >message  
2.</a> It is an example of how a link can trigger an event in  
the same way as a button, using an 'empty' link to the JavaScript  
function as follows: <strong><a  
href="JavaScript:void(0);" ></strong>. <br />  
<input type="button" name="btn3" value="Message 3"  
onclick="showMessage('This is message 3');" />
```

If the same concept is applied to example 1 (which displayed a welcome message incorporating the user's name taken from a text box), it is possible to feed in the text box name as a second argument. The name is then passed to the function along with the message, allowing it to be reused within or between documents.

The original function is:

```
function showAlert() {  
    alert("Hello " + document.egForm1.namebox.value + ".");  
}
```

It is altered by adding argument names between the brackets, and replacing the message and text box reference with these name as follows

```
function showAlert(msg, box) {
```

```
alert(msg + " " + box.value + ".");
}
```

This tells the function to expect the relevant information to be fed in when the function is called.

e.g.

```
<input type="button" name="submit1" value="Submit"
onclick="showAlert('Hello there', this.form.box1);">
```

The first argument is a string which is fed into the function as **msg**. It is placed within single quotation marks rather than double as double quotation marks would interfere with the tag. A comma then follows to separate the arguments.

The second argument uses the **'this'** shorthand to tell the function to take and display the contents from the text box called **'box1'** within the same form as the button that was clicked (**this.form**).

It can thus be called from different boxes in the document and deliver a different message accordingly:

The code in the HTML document is as follows:

```
<form name="egForm2" action="" method="post">
<div class="ques">
<p>What is your name?</p>
<p><input type="text" name="box1" />
<input type="button" name="Submit" value="Submit"
onclick="showAlert('Hello there', this.form.box1);"></p>
<p>What is your full name?</p>
<p><input type="reset" name="reset" value="Reset" /></p>
<input type="text" name="box2" />
<input type="button" name="Submit2" value="Submit"
onclick="showAlert('or perhaps I should call you',
this.form.box2);">
<input type="reset" name="reset2" value="Reset" /></p>
</div>
</form>
```

By placing the script in a file linked to a number of documents, it can also be used across an entire site.

Although this is not very useful for a welcome message script, applying the use of arguments and reusable functions to common tasks such as form validation can be a very efficient way of carrying them out.

Example 2: Simple validation

This example checks that a value has been entered in the text box, preventing submission until this is the case. If a value has been entered a welcome message is displayed.

If this were the entire page, the HTML and JavaScript would be as follows:

```
<html>
<head>
<link href="../../../generic/main.css" rel="stylesheet"
type="text/css">
<title>Simple validation</title>
<script language="javascript" type="text/javascript">
function validateForm(form) {
if (form.namebox.value == ""){
alert("please enter your name.");
form.namebox.focus();
return false;
}
else {
alert("Welcome " + form.namebox.value + ".");
return true;
}
}
</script>
</head>
<body>
<div class="ques">
<form name="egForm2" action="" method="post" onSubmit=" return
validateForm(this);">
<p>What is your name?</p>
<p> <input type="text" name="namebox" />
<input type="submit" name="submit" value="Submit" />
<input type="reset" name="reset" value="Reset" /></p>
</form></div>
</body>
</html>
```

The HTML sections of the document are almost the same as those in example 1. Again, there is a head section that includes a title and a link to a Cascading Style Sheet called "mainstyle.css". The body contains a form which is styled using `<div class></div>` tags. These apply a style called "ques" which is defined in the Style Sheet to everything between the tags. The form contains three input elements - a text box called "namebox", a button called "submit" with "Submit" as the text (the value), and a reset button called "reset" with "Reset" as the value.

The function that carries out the action is called `validateForm()`. It is placed between `<script></script>` tags and is called by the `onSubmit` event handler when the participant submits the form. The `onSubmit` handler is specifically designed for form validation. It is followed by the key word `return` which means that the results of the

validateForm() function will be returned to the piece of code. These results must be either **'true'** (the submission proceeds), or **'false'** (the submission is blocked).

The **onSubmit** handler includes an argument which is the reference to the form (**this**). This feeds the full reference to the form name (**document.egForm2**) into the function when it is triggered. The function then checks whether the box is empty, and displays an alert message accordingly. It also returns the required **'true'** or **'false'** value that will allow or block the submit action. Details about the conditional section of the function (**if/else**) are given below.

Conditionals

Conditionals are integral to validation and, indeed, to programming of all kinds. They allow the code to test whether or not certain conditions have been met and to perform different actions accordingly.

The basic structure of conditionals in JavaScript is as follows:

```
if(Condition A){
Perform an action;
}
else if(condition B){
Perform an alternative action;
}
else {
Perform an alternative action in any other situation;
}
```

Like functions, the actions that take place if a condition is met are placed within curly brackets **'{}'**.

The conditions that the code tests for are placed within brackets **'()''**.

The function in the example **'validateForm()'** basically consists of a conditional to test whether the text box is empty.

```
if (form.namebox.value == "")
```

The **'=='** symbol is an operator meaning 'is equal to', so the conditional tests whether the value is equal to an empty string (**""**).

If this is the case, the following three actions are carried out:

```
{
alert("please enter your name.");
form.namebox.focus();
return false;
}
```

1. An alert box delivers a 'please enter your name' message.

2. The text box is given the focus through the **'focus()'** method (i.e. the cursor is placed inside the text box ready for it to be filled in).
3. **'False'** is returned to the **onSubmit** attribute in the form tag preventing the form submission from proceeding.

If it is not the case, the action is not carried out and the code moves on to the **'else'** statement which performs the following actions if the value is equal to anything else except for an empty string (""):

```
else {
alert("Welcome " + form.namebox.value + ".");
return true;
}
```

1. An alert box delivers a welcome message consisting of 'Welcome ' and the contents of the text box.
2. **'True'** is returned to the **onSubmit** attribute in the form tag allowing the form submission to proceed.

The else statement does not test for specific conditions, but provides a means of carrying out an action if any other condition is met but those specified.

In this case, it could be replaced with an 'else if' statement to test whether the box is not empty:

```
else if (form.namebox.value != "")
```

The **'!='** symbol is an operator meaning 'is not equal to', so the conditional tests whether the value is not equal to an empty string (""). Other common operators used in conditionals are as follows:

Operator	Meaning
x > y	x is greater than y
x < y	x is less than y
x >= y	x is greater than or equal to y
x <= y	x is less than or equal to y

Multiple conditions can also be tested for using the following operators:

Operator	Meaning
condition 1 && condition 2	condition 1 and condition 2
condition 1 condition 2	condition 1 or condition 2

Thus the following conditional would display an alert box message and prevent submission of the form if a text box called **'agebox'** for a question about age is left blank or does not contain a number.

```
if (form.agebox.value == "" || isNaN(form.agebox.value) == true){
alert("Please enter your age as a number.");
return false;
}
```

isNaN(), tests to see whether a value in the brackets **is Not a Number**. The condition will be true if letters or any other characters are contained in the box.

Form validation

Introduction

Using JavaScript, it is possible to check that data is in a suitable format, that any required questions have been answered, and that no questions or selections have been accidentally missed prior to submission. This page will provide an overview of the procedure for adding validation to check the data from common form elements. It is recommended that you review the 'Introduction to JavaScript' page and the section on 'simple validation' in particular before working through this page.

When deciding whether or not to add validation to a questionnaire, it is important to consider the effects it may have. Care must be taken when requiring answers to questions that participants may not wish to answer, as this is likely to lead to drop out. Requiring answers should generally only be considered where it is essential to the success of the questionnaire.

Rather than preventing submission of a questionnaire with incomplete answers, it may be appropriate to prompt the participant to check unanswered fields, giving the option of proceeding with the submission or going back to check. This may increase responses and is certainly more likely to deal with accidental omissions without leading to problems of requiring responses.

Client-side scripting via JavaScript has a great advantages over server-side processing when performing validation as it allows an instant response to any problems with the form submission. It is not necessary for the form information to be submitted to the server and then sent back; the processes happen on the user's computer.

However, It is also important to bear in mind that JavaScript may reduce the accessibility of the questionnaire. Users of text-only or screen-reading browsers are likely to be unable to access any functionality provided by JavaScript, and it is also possible that it may not be fully supported in the user's browser or that he or she may have chosen to deactivate it. For this reason, it is important to ensure that the use of the questionnaire does not depend on JavaScript, and to 'back up' any validation via JavaScript with server-side validation once the form is submitted. This is likely to enhance the security as well as the accessibility of the questionnaire. For more information, see the 'accessibility' and 'server-side validation' sections below.

Example of 'once-only' text box validation

The following is an example of a web form with simple validation applied to a text box. It prevents submission of the form unless 'M,' 'm,' 'F' or 'f' is entered for the question.

It is designed to allow submission the second time the submit button is pressed regardless of whether or not the question has been answered appropriately.

What is your gender (M or F)?

The HTML and JavaScript behind the form is as follows:

The body

```
<body>
<form name="genderCheck" action="" method="post" onSubmit="return
checkForm(this);">
<p>What is your gender (M or F)?</p>
<p><input type="text" name="genderbox" size="1" maxlength="1"
/></p>
<p><input type="submit" name="Submit" value="Submit" /></p>
</form>
</body>
```

Learning activity

Check your understanding of how the form triggers and responds to the validation routines by answering the following questions and referring to the suggested answers below.

Review 'Simple validation' in the 'Introduction to JavaScript' section if necessary.

1. When is the validation triggered?
2. What is the name of the function that performs the validation?
3. Two valid values can be returned to the form tag following validation routines. What are these values and what effect do they have?

Suggested answers

1. The validation is triggered when the user submits the form using the **onSubmit** event handler.
2. The function that performs the validation is called **checkForm** and it feeds the name and reference to the form into the function through the syntax (**this**).
3. The two valid values that can be returned from the validation routine are **true** and **false**. If **true** is returned by the function the submission proceeds. If **false** is returned it is halted.

The validation script

The script makes use of a variable (initially set to **false**) which is used to check whether or not the user has attempted to submit the form previously (it is changed to **true** on the first submission).

If this is the case (or if the form has been completed correctly the first time the submission is attempted) the submission is allowed to proceed (**true** is returned to the form).

Otherwise, the submission is blocked (**false** is returned to the form) and the variable is changed to **true** to allow it to proceed next time.

The script is shown below with comments explaining each section of code.

It is common to add these comments to JavaScript code to provide an explanation of how it works. They are ignored by the browser, but can be viewed in the source code of the HTML

document or the JavaScript file. Comments in JavaScript should be prefixed by two forward-slashes (//) to ensure that the browser ignores them.

Check your understanding of the code by reading the comments. Review the 'Introduction to JavaScript' section if necessary.

```
<script language="javascript" type="text/javascript">
// Set variable for whether submit has been pressed (Placed
outside the function so that its value will not be set to false
every time the function is called, but will be 'remembered'

var submitPressed = false;

// start function and tell it to expect the name and reference to
the form to be passed into it (to replace the word 'form') when
the function is called

function checkForm(form) {
// Check the submission - If the question has been answered
correctly (the box contains 'm', 'M', 'f', or 'F') or if the user
has already tried to submit once before (genderSubmitPressed is
'true'), provide the user with a thank-you alert and return
'true' to the form to allow submission to proceed.

if (submitPressed == true || form.genderbox.value == "m" ||
form.genderbox.value == "M" || form.genderbox.value == "f" ||
form.genderbox.value == "F") {
alert("Thank you for completing the form.");
return true;
}

// In any other situation, deliver an alert box message to the
user reminding them to answer the question. Change the
submitPressed variable to 'true' to allow submission to proceed
next time and return 'false' to the form to prevent submission
this time.

else {
alert("Please insert your gender which is needed for analysis
purposes and which will not compromise the anonymity of your
data. If you would rather not provide this information, press the
submit button again to proceed.");
submitPressed = true;
return false;
}

// end function
}
</script>
```

In a situation where the researcher wishes to require an answer to the question and not allow submission at the second attempt, this could easily be done by removing the three references

to the 'submitPressed' variable in the code (`var submitPressed = false;`, `submitPressed == true` ||, and `submitPressed = true;`).

Radio buttons

It is also possible to check that one of a group of radio buttons has been checked as in the following form.

How often do you use the internet?

everyday

2-3 days per week

4-5 days per week

6-7 days per week

less than once a week

This is made possible by the fact that all of the radio buttons have the same name and are thus automatically included in an array. This means that each of the buttons is given a number starting with zero. These numbers can be used to refer to each Check box using the following syntax:

form.often[x]

where **x** is the number of the button and where **form** is replaced by the name and reference to the form (e.g. `window.document.formname`) which is passed into the function when it is called.

The HTML for the radio buttons is as follows:

```
<input type="radio" name="often" value="everyday" />
everyday<br />
<input type="radio" name="often" value="2-3 days per week" />
2-3 days per week<br />
<input type="radio" name="often" value="4-5 days per week" />
4-5 days per week<br />
<input type="radio" name="often" value="6-7 days per week" />
6-7 days per week<br />
<input type="radio" name="often" value="less than once a week" />
less than once a week </p>
```

The reference to the first button is:

form.often[0]

while the reference to the last (the fifth) is:

form.often[4]

Each radio button has a property **checked** which has a value **true** or **false**. This can be used to ascertain whether it has been selected (**checked is true**) or has not been selected (**checked is false**).

The section of the validation script that checks whether one of the buttons has been selected is as follows:

```
if (form.often[0].checked == true || form.often[1].checked ==
true || form.often[2].checked == true || form.often[3].checked ==
true || form.often[4].checked == true){
alert("Thank you for completing the form.");
return true;
}
```

This displays the thank-you message and returns true to the form to allow submission to proceed in the following circumstances:

circumstances	code
button 1 has been selected:	<code>form.often[0].checked == true</code>
or:	<code> </code>
button 2 has been selected:	<code>form.often[1].checked == true</code>
or:	<code> </code>
button 3 has been selected:	<code>form.often[2].checked == true</code>
or:	<code> </code>
button 4 has been selected:	<code>form.often[3].checked == true</code>
or:	<code> </code>
button 5 has been selected:	<code>form.often[4].checked == true</code>

This can be simplified by looping through each of the boxes to check whether or not they have been selected, as follows:

```
for (var i = 0; i<form.often.length; i++){
if (form.often[i].checked == true){
questionAnswered = true;
}
}
if (questionAnswered == true){
alert("Thank you for completing the form.");
return true;
}
```

This begins a loop which feeds the value of `i` into the conditional (`if (form.often[i].checked == true)`)

The initial value of `i` is set to 0 (`var i = 0;`) and it is set to increase by one at the end of each loop (`i++`) until `i` is no longer less than the length of the array of Check boxes called `often` (`i<form.often.length;`) which in this case is five (there are five boxes).

This means that the conditional (`if (form.often[i].checked == true)`) is carried out five times with `i` replaced with 0, 1, 2, 3, and 4.

In each case, if the button has been selected, a variable `questionAnswered` is set to `true`.

This is followed by another conditional which displays the thank-you message and returns true to the form if `questionAnswered` is `true`.

The complete code is as follows:

```
<script language="javascript" type="text/javascript">  
// Set variable for whether submit has been pressed  
var submitPressed = false;  
// start function  
function checkForm(form) {  
// Set variable for whether question has been answered  
var questionAnswered = false;  
// Check the submission - If any of the buttons have been  
selected, change 'questionAnswered' variable to 'true'.  
for (var i = 0; i<form.often.length; i++){  
if (form.often[i].checked == true){  
questionAnswered = true;  
}  
}  
// If 'submitPressed' is 'true' (the submit button has already  
been pressed) or 'questionAnswered' is 'true' (an option has been  
selected), provide the user with a thank-you alert and return  
'true' to the form to allow submission to proceed. .  
if (submitPressed == true || questionAnswered == true){  
alert("Thank you for completing the form.");  
return true;  
}  
// In any other situation, deliver an alert box message to the  
user reminding them to answer the question. Change the  
submitPressed variable to 'true' to allow submission to proceed  
next time and return 'false' to the form to prevent submission  
this time.  
else {  
alert("Please select an answer. If you would rather not provide  
this information, press the submit button again to proceed.");  
submitPressed = true;  
return false;  
}  
// end function  
}  
</script>
```

Check boxes

Like radio buttons, a group of check boxes can be given the same name. In this case, they can be validated in the same way as radio buttons using the array which automatically numbers the boxes.

Which of the following do you regularly use the internet for?

- E mail
- Finding information about things to buy
- Making purchases
- Entertainment
- Finding general information
- Educational courses
- Downloading music
- Discussion boards
- Real-time chat

The HTML for the Check boxes is as follows:

```
<p>Which of the following do you regularly use the internet
for?</p>
<p>
<input type="Check box" name="uses" value="email" />
E mail<br />
<input type="Check box" name="uses" value="finding information
about things to buy" />
Finding information about things to buy<br />
<input type="Check box" name="uses" value="making purchases" />
Making purchases<br />
<input type="Check box" name="uses" value="entertainment" />
Entertainment<br />
<input type="Check box" name="uses" value="finding general
information" />
Finding general information<br />
<input type="Check box" name="uses" value="educational courses"
/>
Educational courses<br />
<input type="Check box" name="uses" value="downloading music" />
Downloading music<br />
<input type="Check box" name="uses" value="discussion boards" />
Discussion boards<br />
<input type="Check box" name="uses" value="real-time chat" />
Real-time chat</p>
<p>
```

The Check boxes are given the same name (**uses**), but different values. When the form is submitted, the value of any checked box can be added to a variable. Each of the different

values is added separated by a comma. Thus, for example, if the first three Check boxes are selected, the variable will be :

```
"email, finding information about things to buy, making purchases"
```

The code for checking that an option has been selected is exactly the same as that for the radio buttons, except that the name given to the Check boxes is added to the loop and conditionals, as follows:

```
for (var i = 0; i<form.uses.length; i++){  
  if (form.uses[i].checked == true){  
    questionAnswered = true;  
  }  
}
```

This is a key advantage of using the loop to check each of the radio buttons or check boxes rather than including a conditional for every button or box. The same piece of code can be used for each group regardless of the number of boxes, with the only necessary modification being the name.

Select boxes

Select boxes can be validated in a similar way to radio buttons and Check boxes, as the options in a box are also counted in an array.

How would you rate your skill as an internet user?

The HTML for the select box is as follows:

```
<p>How would you rate your skill as an internet user?</p>  
<p>  
<select name="skill">  
<option value="not answered">Select an option</option>  
<option value="not answered">-----</option>  
<option value="very advanced">Very advanced</option>  
<option value="advanced">Advanced</option>  
<option value="average">Average</option>  
<option value="basic">Basic</option>  
<option value="very basic">Very basic</option>  
</select>  
</p>  
<p>
```

It is given the name **skill** and each option is given an appropriate value. The options are counted in an array as follows:

```
form.skill.options[x]
```

where **x** is the number of the option (starting from 0) and where **form** is replaced by the name and reference to the form (e.g. **window.document.formname**) which is passed into the function when it is called.

Thus, the reference to the first option (labeled 'Select an option' and given the value 'not answered') is:

```
form.skill.options[0]
```

while the reference to the last (the seventh, labeled 'Very basic' and given the value 'very basic') is:

```
form.skill.options[6]
```

The first option provides the instruction to the user (**Select an option**), and the second acts as a divider between this and the options proper(-----). Thus, it can be assumed that if either of these options is selected, the question has not been answered.

It is therefore possible to check that a valid option has been selected through the following code:

```
// Check the submission - If both the first and second options  
have not been selected, provide the user with a thank-you alert  
and return 'true' to the form to allow submission to proceed.
```

```
if (form.skill.options[0].selected != true &&  
form.skill.options[1].selected != true) {  
  
alert("Thank you for completing the form.");  
return true;  
  
}
```

Validating multiple form elements

As with the previous examples, the following questionnaire prevents the first submission of the form unless all the questions have been answered and allows submission the second time the submit button is pressed regardless of whether or not all the questions have been answered appropriately.

In this case, however, there is more than one form element, so it is helpful to inform the user of which question(s) need attention. This is done through delivery of an alert-box message when a problem is found.

1. What is your name?

2. How often do you use the internet?

everyday

2-3 days per week

4-5 days per week

6-7 days per week

less than once a week

This is done through the use of extra variables which are set to record which questions have not been answered and to add the question numbers to the alert box message.

The code for this is shown in the following learning activity.

Learning activity

This activity will allow you to check your understanding of how the code works, by matching the comments to the sections of code that they describe. It will also act as a review of once-only validation of textboxes and radio buttons.

Match the pieces of code (A-H) to the comments that describe them (1-8).

Code

```
<script language="javascript" type="text/javascript">
```

```
A  
var submitPressed = false;  
var q1Answered = false;  
var q2Answered = false;  
var alertMessage = "";
```

```
// start function  
function checkForm(form) {
```

```
B  
  
else{  
q1Answered = true;  
}
```

```
C  
  
for (var i = 0; i<form.often.length; i++){  
if (form.often[i].checked == true){  
q2Answered = true;  
}  
}
```

```
D  
  
if (q2Answered == false){  
alertMessage = alertMessage + "2 ";  
}
```

```
E  
  
if (submitPressed == true || (q1Answered == true && q2Answered == true)){
```

```
alert("Thank you for completing the form.");
return true;
}
```

F

```
else {
alert("Please check your answers to the following
question(s):\n\r" + alertMessage + "\n\rIf you are happy with
them, press the submit button again to proceed with the
submission.");
}
```

G

```
submitPressed = true;
return false;
}
// end function
}
</script>
```

Comments

1. // Check question 1 - If the box is empty, the question has not been answered. In this case, add the question number to the message that will be delivered to the user to indicate that there are problems with certain questions.
2. // If both questions have been answered (q1Answered and q2Answered are both true) or if the user has already tried to submit once before (submitPressed is true), provide the user with a thank-you alert and return 'true' to the form to allow submission to proceed.
3. // If either or both of the questions has not been answered, deliver an alert box message to the user and include the variable 'alertMessage' which was used to collect the numbers of the unanswered questions.
4. // Change the submitPressed variable to true to allow submission to proceed next time and return 'false' to the form to prevent submission this time.
5. // If 'q2Answered' is still false, the question has not been answered, so add the question number to the message.
6. // If the box is not empty, the question has been answered, so the 'q1Answered' variable is changed to 'true'.
7. // Check question 2 - If any of the buttons have been selected, change 'q2Answered' variable to 'true'.
8. // Set variables
// Variable for whether submit has been pressed
// Variable for whether each question has been answered
// Variable for the final alert box message.

Answers

A=8
B=1
C=6
D=7
E=5
F=2
G=3
H=4

Complete code:

```
<script language="javascript" type="text/javascript">  
  
// Set variables  
// Variable for whether submit has been pressed  
// Variable for whether each question has been answered  
// Variable for the final alert box message.  
  
var submitPressed = false;  
var q1Answered = false;  
var q2Answered = false;  
var alertMessage = "";  
  
// start function  
  
function checkForm(form) {  
  
// Check question 1 - If the box is empty, the question has not  
// been answered. In this case, add the question number to the  
// message that will be delivered to the user to indicate that there  
// are problems with certain questions.  
  
if (form.namebox.value == ""){  
  
alertMessage = "1 ";  
  
}  
  
// If the box is not empty, the question has been answered, so  
// the 'q1Answered' variable is changed to 'true'.  
  
else{  
  
q1Answered = true;  
  
}  
  
// Check question 2 - If any of the buttons have been selected,  
// change 'q2Answered' variable to 'true'.  
  
for (var i = 0; i<form.often.length; i++){  
  
if (form.often[i].checked == true){  
  
q2Answered = true;  
  
}  
  
}
```

```

}
// If 'q2Answered' is still false, the question has not been
answered, so add the question number to the message.
if (q2Answered == false){
alertMessage = alertMessage + "2 ";
}
// If both questions have been answered (q1Answered and
q2Answered are both true) or if the user has already tried to
submit once before (submitPressed is true), provide the user with
a thank-you alert and return 'true' to the form to allow
submission to proceed.
if (submitPressed == true || (q1Answered == true && q2Answered ==
true)){
alert("Thank you for completing the form.");
return true;
}
// If either or both of the questions has not been answered,
deliver an alert box message to the user and include the variable
'alertMessage' which was used to collect the numbers of the
unanswered questions.
else {
alert("Please check your answers to the following
question(s):\n\r" + alertMessage + "\n\rIf you are happy with
them, press the submit button again to proceed with the
submission.");
}
// Change the submitPressed variable to true to allow submission
to proceed next time and return 'false' to the form to prevent
submission this time.
submitPressed = true;
return false;
}
// end function
}
</script>

```

Summary of the working of the code

The text box and radio button validation routines shown in the sections above are carried out, but where a question has been answered a variable for that question is set to true rather than the submission occurring immediately. Where a question has not been answered, the

submission is not immediately blocked, but the number of that question is added to a variable which is then incorporated into an alert-box message.

Finally, both the question variables are checked and where they are found to be true (i.e. they have been answered) or where the form has been submitted once before (the form submission variable is found to be true) a thank-you message is delivered. If any question variable is false, the alert-box message is delivered and the form submission variable is changed to true.

Putting it all together

This section describes examples of validation of a questionnaire with multiple questions and question types. The complete code for each method shown can be viewed in the 'Adapting the code' section below.

The following questionnaire about internet use incorporates all of the different validation elements outlined above:

1. What is your name?

2. How old are you?

 years

3. What is your gender (M or F)?

4. How often do you use the internet?

everyday

2-3 days per week

4-5 days per week

6-7 days per week

less than once a week

5. Which of the following do you regularly use the internet for?

(You can select as many options as you like. If you would like to remove a selection you have made, select it again to deselect it).

E mail

Finding information about things to buy

Making purchases

Entertainment

Finding general information

Educational courses

Downloading music

Discussion boards

Real-time chat

6. How would you rate your skill as an internet user?

Select an option ▼

7. What, in your opinion, are the three main advantages of the internet?

A potential problem with the way the questionnaire is validated is that a single message is delivered to the user which includes all the questions requiring attention. This is not ideal as the user is unlikely to be able to remember which questions need to be checked in cases where there is more than one or two. Two more complex possible alternatives are shown below.

Alternative 1: One-by-one validation

In this case, each question is checked individually. If a question has not been answered or previously submitted, an alert message is displayed to the participant, and false is returned to the form to prevent submission. A variable for that question is also set to true so that submission will not be blocked if the participant resubmits without answering the question for a second time.

If the first question has been answered or has been submitted before, the second is checked. If this has been answered or previously submitted, the third is checked. This continues until all have been checked and a thank-you message is then delivered.

A potential problem with the way this questionnaire is validated is that it tends to overuse alert messages. Where a number of questions have not been answered, the participant may become frustrated by this, which could lead to increased drop-out rates. However, it is no more difficult to implement and adapt for different questionnaires than the first example.

Alternative 2: Displaying messages within the questionnaire

This alternative is similar to the first example in that all the questions are checked together. The questionnaire prevents the first submission of the form unless all the questions have been answered and allows submission the second time the submit button is pressed regardless of which questions may not have been answered appropriately.

Rather than delivering an alert-box message with the numbers of the questions that have not been answered correctly, however, a message is revealed next to the questions on the questionnaire itself.

This probably provides the most effective validation in terms of usability, but there may be problems for participants using older browsers (with version numbers lower than Netscape Navigator 5 or Internet Explorer 4) who may not be able to access the messages.

For this reason, an alert box message is included indicating that there is a problem with the answers. For users with a browser capable of displaying them, this is backed up with the messages indicating where the problems are. However, those who cannot access these

messages are still informed of the reason for the failed submission, prompted to check their answers, and given the message that pressing the submit button a second time will allow it to proceed.

Thus, while the use of the messages adds to the functionality of the questionnaire, it does not prevent it from being usable for those who are unable to access it (see the 'Accessibility' section).

The messages are revealed using `<div>` and `` tags in the HTML code with CSS and JavaScript. Implementing and adapting this method of presenting the results of validation is slightly more involved than the previous examples, but the validation routines are very similar to those of the first example.

Adapting the code

The complete HTML and JavaScript for each of the three questionnaires shown in this section can be seen below. The code can be saved, or copied and pasted into your text editor for adaptation and use in questionnaires with different types and numbers of questions. There are explanatory comments in the code and some general notes on adaptation are as follows:

1. When removing questions, make sure that associated code and variables are also removed.
2. When adding validation to new questions, be sure to copy in full the associated code and variables from a question of the same type.
3. When adding form elements for new questions, give them a unique name and make sure that this is added correctly into the code copied from similar questions.
4. Similarly, make sure that new variable names are unique and added correctly into all relevant parts of the code.
5. Ensure that you have included the required punctuation, or the code may not run correctly. For example, make sure that functions are contained within curly brackets (`{}`), and that conditionals are within brackets, with actions resulting from conditionals in curly brackets. See the 'Introduction to JavaScript' section of this guide for further information if required.

Questionnaire 1: Alert-box message

```
<html>
<head>
<title>Exploring ORMs | Example of alert-box validation</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-
8859-1">
<style type="text/css">
body {
font-family: Verdana, Arial, Helvetica, sans-serif;
font-size: 75%;
color: #003;
}
p {
font-size: 100%;
margin: 10px 0px;
}
.form {
background-color: #ffffec;
border: #003 1px solid;
```



```

padding: 10px;
}
</style>
<script language="javascript" type="text/javascript">

// Set variables

// Variable for whether submit has been pressed. Delete this if
you prefer to require an answer to the questions and not allow
submission at the second attempt.

var submitPressed = false;

// Variable for each question.

var q1Answered = false;
var q2Answered = false;
var q3Answered = false;
var q4Answered = false;
var q5Answered = false;
var q6Answered = false;
var q7Answered = false;

// Variable for the final alert box message

var alertMessage = "";

// start function

function checkForm(form) {

// Check question 1 - If the box is empty, the question has not
been answered. In this case, add the question number to the
message that will be delivered to the user to indicate that there
are problems with certain questions.

if (form.namebox.value == ""){
alertMessage = "1 ";
}

// If the box is not empty, the question has been answered, so
the 'q1Answered' variable is changed to 'true'.

else{
q1Answered = true;
}

// Check question 2 - If the box is empty or it does not contain
a number, the question has been not been answered. In this case,
add the number to the message.

if (form.agebox.value == "" || isNaN(form.agebox.value) == true){
alertMessage = alertMessage + "2 ";
}
}

```

```

// In any other situation, assume the question has been answered,
so change the 'q2Answered' variable to 'true'.

else{
q2Answered = true;
}

// Check question 3 - If the box does not contain 'm', 'M', 'f',
or 'F', the question has been not been answered correctly. In
this case, add the number to the message.

if (form.genderbox.value != "m" && form.genderbox.value != "M" &&
form.genderbox.value != "f" && form.genderbox.value != "F" ){
alertMessage = alertMessage + "3 ";
}

// In any other situation, the question has been answered, so
change the 'q3Answered' variable to 'true'.

else{
q3Answered = true;
}

// Check question 4 - Check each of the radio buttons. If one has
been selected, the question has been answered, so change the
'q4Answered' variable to 'true'.

for (var i = 0; i<form.often.length; i++){
if (form.often[i].checked == true){
q4Answered = true;
}
}

// If the 'q4Answered' variable is still 'false', the question
has not been answered so add the number to the message.

if (q4Answered == false){
alertMessage = alertMessage + "4 ";
}

// Check question 5 - If one of the Check boxes has been
selected, the question has been answered, so change the
'q4Answered' variable to 'true'.

for (var j = 0; j<form.uses.length; j++){
if (form.uses[j].checked == true){
q5Answered = true;
}
}

// If the 'q5Answered' variable is still 'false', the question
has not been answered so add the number to the message.

```

```

if (q5Answered == false){
alertMessage = alertMessage + "5 ";
}

// Check question 6 - If one of the first two options has been
selected, the question has not been answered and the number is
added to the message.

if (form.skill.selectedIndex == 0 || form.skill.selectedIndex ==
1){
alertMessage = alertMessage + "6 ";
}

// In any other situation, a valid option has been selected, so
change the 'q6Answered' variable to 'true'.

else{
q6Answered = true;
}

// Check question 7 - If the textarea is empty, the question has
not been answered. In this case, add the question number to the
message.

if (form.advantages.value == ""){
alertMessage = alertMessage + "7 ";
}

// Otherwise, change the 'q7Answered' variable to 'true'.

else{
q7Answered = true;
}

// If all the questions have been answered (q1Answered to
q7Answered are all true) or if the user has already tried to
submit once before (submitPressed is true), provide the user with
a thank-you alert and return 'true' to the form to allow
submission to proceed. Delete '|| submitPressed == true' if you
prefer to require an answer to the questions and not allow
submission at the second attempt.

if (q1Answered == true && q2Answered == true && q3Answered ==
true && q4Answered == true && q5Answered == true && q6Answered ==
true && q7Answered == true || submitPressed == true){
alert("Thank you for completing the form.");
return true;
}

// If some questions have not been answered, deliver an alert box
message to the user and include the variable 'alertMessage' which
was used to collect the numbers of the unanswered questions.
Change the submitPressed variable to 'true' to allow submission
to proceed next time and return 'false' to the form to prevent

```

submission this time. Delete 'submitPressed = true;' if you prefer to require an answer to the questions and not allow submission at the second attempt.

```
else {
alert("Please check your answers to the following questions:\n\r"
+ alertMessage + "\n\rIf you are happy with them, press the
submit button again to proceed with the submission.");
submitPressed = true;
return false;
}
}
```

```
</script>
```

```
</head>
```

```
<body>
```

```
<div class="form">
```

```
<form name="exampleform1" action="" method="post"
```

```
onSubmit="return checkForm(this);">
```

```
<p> 1. What is your name? </p>
```

```
<p><input type="text" name="namebox" /></p>
```

```
<p>2. How old are you?</p>
```

```
<p><input name="agebox" type="text" size="5" maxlength="3" />
years</p>
```

```
<p>3. What is your gender (M or F)?</p>
```

```
<p><input name="genderbox" type="text" size="1" maxlength="1"
/></p>
```

```
<p>4. How often do you use the internet?</p>
```

```
<p>
```

```
<input type="radio" name="often" value="everyday" /> everyday<br
/>
```

```
<input type="radio" name="often" value="2-3 days per week" /> 2-3
days per week<br />
```

```
<input type="radio" name="often" value="4-5 days per week" /> 4-5
days per week<br />
```

```
<input type="radio" name="often" value="6-7 days per week" /> 6-7
days per week<br />
```

```
<input type="radio" name="often" value="less than once a week" />
less than once a week
```

```
</p>
```

```
<p>5. Which of the following do you regularly use the internet
for? <br />(You can select as many options as you like. If you
would like to remove a selection you have made, select it again
to deselect it).</p>
```

```
<p>
```

```
<input type="Check box" name="uses" value="email" />E mail<br />
```

```
<input type="Check box" name="uses" value="finding information
about things to buy" />Finding information about things to buy<br
/>
```

```
<input type="Check box" name="uses" value="making purchases"
/>Making purchases<br />
```

```
<input type="Check box" name="uses" value="entertainment"
```

```

/>Entertainment<br />
<input type="Check box" name="uses" value="finding general
information" />Finding general information<br />
<input type="Check box" name="uses" value="educational courses"
/>Educational courses<br />
<input type="Check box" name="uses" value="downloading music"
/>Downloading music<br />
<input type="Check box" name="uses" value="discussion boards"
/>Discussion boards<br />
<input type="Check box" name="uses" value="real-time chat" />
Real-time chat</p>
<p>6. How would you rate your skill as an internet user?</p>
<p>
<select name="skill">
<option selected="selected">Select an option</option>
<option>-----</option>
<option value="very advanced">Very advanced</option>
<option value="advanced">Advanced</option>
<option value="average">Average</option>
<option value="basic">Basic</option>
<option value="very basic">Very basic</option>
</select>
</p>
<p>7. What, in your opinion, are the three main advantages of the
internet?</p>
<p><textarea name="advantages" rows="5" cols="50"></textarea></p>
<p><input type="submit" name="Submit" value="Submit" /></p>
</form>
</div>
</body>
</html>

```

Questionnaire 2: One-by-one validation

```

<html>
<head>
<title>Exploring ORMs | Example of one-by-one validation</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-
8859-1">
<link href="../generic/nn4.css" rel="stylesheet" type="text/css">
<style type="text/css">
body {
font-family: Verdana, Arial, Helvetica, sans-serif;
font-size: 75%;
color: #003;
}
p {
font-size: 100%;
margin: 10px 0px;
}
.form {

```

```

background-color: #ffffec;
border: #003 1px solid;
padding: 10px;
}
</style>
<script language="javascript" type="text/javascript">

// Set variables for each question. Delete these if you prefer to
require an answer to the questions and not allow submission at
the second attempt.

var q1Answered= false;
var q2Answered= false;
var q3Answered= false;
var q4Answered= false;
var q5Answered= false;
var q6Answered= false;
var q7Answered= false;

// start function
function checkForm(form) {

// Check question 1 - If the 'q1Answered' variable is false,
check the contents of the box. If it is not false, ignore the
check and move onto the validation for question 2. Delete this if
you prefer to require an answer to the questions and not allow
submission at the second attempt. (And do same to the equivalent
line for the other questions)

if (q1Answered==false){

// If the box is empty, the question has not been answered. In
this case, deliver an alert box message and change the
'q1Answered' variable to 'true' so that it will not prevent
submission next time the submit button is pressed. Return 'false'
to the form to prevent submission this time. Delete 'q1Answered=
true;' if you prefer to require an answer to the questions and
not allow submission at the second attempt. (And do same to the
equivalent line for the other questions)

if (form.namebox.value == ""){
alert ("Please complete question 1 or leave it blank and press
the submit button again to proceed with the submission.");
q1Answered= true;
return false;
}
}

// Check question 2 - If the 'q2Answered' variable is false,
check the contents of the box. If it is not false, ignore the
check and move onto the validation for question 3.

if (q2Answered==false){

```

```

// If the box is empty or it does not contain a number, the
question has not been answered correctly. In this case, deliver
an alert box message and change the 'q2Answered' variable to
'true' so that it will not prevent submission next time the
submit button is pressed. Return 'false' to the form to prevent
submission this time.

if (form.agebox.value == "" || isNaN(form.agebox.value) == true){
alert ("Please enter your age as a number for question 2 or leave
the box blank and press the submit button again to proceed with
the submission.");
q2Answered = true;
return false;
}
}

// Check question 3 - If the 'q3Answered' variable is false,
check the contents of the box. If it is not false, ignore the
check and move onto the validation for question 4.

if (q3Answered==false){

// If the box does not contain 'M', 'm', 'F' or 'f', the question
has not been answered correctly. In this case, deliver an alert
box message and change the 'q3Answered' variable to 'true' so
that it will not prevent submission next time the submit button
is pressed. Return 'false' to the form to prevent submission this
time.

if (form.genderbox.value != "m" && form.genderbox.value != "M" &&
form.genderbox.value != "f" && form.genderbox.value != "F" ){
alert ("Please insert your gender for question 3 or leave the box
blank and press the submit button again to proceed with the
submission.");
q3Answered = true;
return false;
}
}

// Check question 4 - If the 'q4Answered' variable is false,
check the radio buttons. If it is not false, ignore the check and
move onto the validation for question 5.

if (q4Answered==false){

// Set a variable for whether an option has been selected.

var q4Selected = false;

// Check each of the radio buttons. If one has been selected, the
question has been answered, so change the 'q4Selected' variable
to 'true'.

for (var i = 0; i<form.often.length; i++){
if (form.often[i].checked == true){

```

```

q4Selected = true;
}
}

// If the 'q4Selected' variable is still 'false', the question
has not been answered, so deliver an alert box message and change
the 'q4Answered' variable to 'true' so that it will not prevent
submission next time the submit button is pressed. Return 'false'
to the form to prevent submission this time.

if (q4Selected == false){
alert ("Please choose an answer for question 4 or press the
submit button again to proceed with the submission.");
q4Answered = true;
return false;
}
}

// Check question 5 - If the 'q5Answered' variable is false,
check the Check boxes. If it is not false, ignore the check and
move onto the validation for question 6.

if (q5Answered==false){

// Set a variable for whether at least one Check box has been
selected.

var q5Checked = false;

// Check each of the Check boxes. If one has been selected, the
question has been answered, so change the 'q5Selected' variable
to 'true'.

for (var j = 0; j<form.uses.length; j++){
if (form.uses[j].checked == true){
q5Checked = true;
}
}

// If the 'q5Checked' variable is still 'false', the question has
not been answered, so deliver an alert box message and change the
'q5Answered' variable to 'true' so that it will not prevent
submission next time the submit button is pressed. Return 'false'
to the form to prevent submission this time.

if (q5Checked == false){
alert ("You have not selected an answer for question 5. Please
check this and press the submit button again to proceed with the
submission.");
q5Answered = true;
return false;
}
}
}

```



```

// Check question 6 - If the 'q6Answered' variable is false,
check the select box. If it is not false, ignore the check and
move onto the validation for question 7.

if (q6Answered==false){

// Check question 6 - If one of the first two options has been
selected, the question has not been answered. In this case,
deliver an alert box message and change the 'q6Answered' variable
to 'true' so that it will not prevent submission next time the
submit button is pressed. Return 'false' to the form to prevent
submission this time.

if (form.skill.selectedIndex == 0 || form.skill.selectedIndex ==
1){
alert ("You have not selected an answer for question 6. Please
check this and press the submit button again to proceed with the
submission.");
q6Answered = true;
return false;
}
}

// Check question 7 - If the 'q7Answered' variable is false,
check the contents of the textarea. If it is not false, ignore
the check and move onto the thank-you message.

if (q7Answered==false){

// If the textarea is empty, the question has not been answered.
In this case, deliver an alert box message and change the
'q7Answered' variable to 'true' so that it will not prevent
submission next time the submit button is pressed. Return 'false'
to the form to prevent submission this time.

if (form.advantages.value == ""){
alert ("Please complete question 7 or leave it blank and press
the submit button again to proceed with the submission.");
q7Answered= true;
return false;
}
}

// If the code makes it to this stage without returning 'false'
to the form, all the questions have been answered correctly or
have been submitted twice. Deliver a thank-you alert message and
return 'true' to the form to allow submission to proceed.

alert("Thank you for completing the form.");
return true;
}
</script>
</head>
<body>

```

```

<div class="form">
<form name="exampleform1" action="" method="post"
onSubmit="return checkForm(this);">
<p> 1. What is your name? </p>
<p><input type="text" name="namebox" /></p>
<p>2. How old are you?</p>
<p><input name="agebox" type="text" size="5" maxlength="3" />
years</p>
<p>3. What is your gender (M or F)?</p>
<p><input name="genderbox" type="text" size="1" maxlength="1"
/></p>
<p>4. How often do you use the internet?</p>
<p>
<input type="radio" name="often" value="everyday" /> everyday<br
/>
<input type="radio" name="often" value="2-3 days per week" /> 2-3
days per week<br />
<input type="radio" name="often" value="4-5 days per week" /> 4-5
days per week<br />
<input type="radio" name="often" value="6-7 days per week" /> 6-7
days per week<br />
<input type="radio" name="often" value="less than once a week" />
less than once a week
</p>
<p>5. Which of the following do you regularly use the internet
for? <br />
(You can select as many options as you like. If you would like to
remove a selection you have made, select it again to deselect
it).</p>
<p>
<input type="Check box" name="uses" value="email" /> E mail<br />
<input type="Check box" name="uses" value="finding information
about things to buy" />Finding information about things to buy<br
/>
<input type="Check box" name="uses" value="making purchases"
/>Making purchases<br /><input type="Check box" name="uses"
value="entertainment" />Entertainment<br />
<input type="Check box" name="uses" value="finding general
information" />Finding general information<br />
<input type="Check box" name="uses" value="educational courses"
/>Educational courses<br />
<input type="Check box" name="uses" value="downloading music"
/>Downloading music<br />
<input type="Check box" name="uses" value="discussion boards"
/>Discussion boards<br />
<input type="Check box" name="uses" value="real-time chat"
/>Real-time chat</p>
<p>6. How would you rate your skill as an internet user?</p>
<p>
<select name="skill">
<option selected="selected">Select an option</option>

```

```

<option>-----</option>
<option value="very advanced">Very advanced</option>
<option value="advanced">Advanced</option>
<option value="average">Average</option>
<option value="basic">Basic</option>
<option value="very basic">Very basic</option>
</select>
</p>
<p>7. What, in your opinion, are the three main advantages of the
internet?</p>
<p><textarea name="advantages" rows="5" cols="50"></textarea></p>
<p><input type="submit" name="Submit" value="Submit" /></p>
</form>
</div>
</body>
</html>

```

Questionnaire 3: Validation via messages within the questionnaire

```

<html>
<head>
<title>Exploring ORMs | Example of validation via messages in the
questionnaire</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-
8859-1">
<link href="../generic/nn4.css" rel="stylesheet" type="text/css">
<style type="text/css">
body {
font-family: Verdana, Arial, Helvetica, sans-serif;
font-size: 75%;
color: #003;
}
p {
font-size: 100%;
margin: 10px 0px;
}
.form {
background-color: #ffffec;
border: #003 1px solid;
padding: 10px;
}
.response {
font-weight: bold;
color: #f00;
}
</style>
<script language="javascript" type="text/javascript">

// The messages to the participant are delivered by showing or
hiding layers. This is done differently according to the type of

```

browser the participant is using. This block of code checks the type of browser

```
var isIE4 = false;
var isCompliant = false;
if(document.getElementById)
{
if(!document.all)
{
isCompliant=true;
}
if(document.all)
{
isIE4=true;
}
}
```

// Code to allow the hidden layers used to deliver the messages to be made visible. A different method is used according to the type of browser identified in the previous block of code.

```
function aLs(layerID)
{
var returnLayer = "null";
if(isIE4)
{
returnLayer = eval("document.all." + layerID + ".style");
}
if(isCompliant)
{
returnLayer = eval("document.getElementById('" + layerID +
"').style");
}
return returnLayer;
}
```

// Function to make a layer visible when called. The layer name is fed into the function when it is called.

```
function showResponse(ID)
{
aLs(ID).display = "";
}
```

// Function to make a layer invisible when called. The layer name is fed into the function when it is called.

```
function hideResponse(ID)
{
aLs(ID).display = "none";
}
```

// Variable for whether submit has been pressed

```

var submitPressed = false;

// Variable for each question.

var q1Answered = false;
var q2Answered = false;
var q3Answered = false;
var q4Answered = false;
var q5Answered = false;
var q6Answered = false;
var q7Answered = false;

// start validation function

function checkForm(form) {

// Check each question in turn and change the associated variable
to 'true' if they have been answered. (See comments for the first
example questionnaire and/or the appropriate sections of the
'Form validation' page for explanations if required.

if (form.namebox.value != ""){
q1Answered = true;
}
if (form.agebox.value != "" && isNaN(form.agebox.value) != true){
q2Answered = true;
}
if (form.genderbox.value == "m" || form.genderbox.value == "M" ||
form.genderbox.value == "f" || form.genderbox.value == "F" ){
q3Answered = true;
}
for (var i = 0; i<form.often.length; i++){
if (form.often[i].checked == true){
q4Answered = true;
}
}
for (var j = 0; j<form.uses.length; j++){
if (form.uses[j].checked == true){
q5Answered = true;
}
}
if (form.skill.selectedIndex != 0 && form.skill.selectedIndex !=
1){
q6Answered = true;
}
if (form.advantages.value != ""){
q7Answered = true;
}
}

// If all the questions have been answered (q1Answered to
q7Answered are all true) or if the user has already tried to
submit once before (submitPressed is true), call the
'hideResponse' function for each of the message layers to make

```

sure that, if they have previously been shown, they are hidden again. Provide the user with a thank-you alert and return 'true' to the form to allow submission to proceed.

```
if (q1Answered == true && q2Answered == true && q3Answered ==
true && q4Answered == true && q5Answered == true && q6Answered ==
true && q7Answered == true || submitPressed == true){
hideResponse('incorrect');
hideResponse('incorrect1');
hideResponse('incorrect2');
hideResponse('incorrect3');
hideResponse('incorrect4');
hideResponse('incorrect5');
hideResponse('incorrect6');
hideResponse('incorrect7');
alert("Thank you for completing the form.");
return true;
}
```

```
// If some questions have not been answered, deliver an alert box
message to the user
```

```
else {
alert("There appears to be problems with your answers. Please
check and if you are happy with your answers, press the submit
button again to proceed with the submission.");
```

```
// Show the layer called 'incorrect' which contains the
instructions to the participant (See HTML body)
```

```
showResponse('incorrect');
```

```
// Check the first question. If it has not been answered (its
variable is still false), show the layer which shows there is a
problem with that question (called 'incorrect1 - see HTML body)
```

```
if (q1Answered == false){
showResponse('incorrect1');
}
```

```
// Do the same check for the other questions.
```

```
if (q2Answered == false){
showResponse('incorrect2');
}
```

```
if (q3Answered == false){
showResponse('incorrect3');
}
```

```
if (q4Answered == false){
showResponse('incorrect4');
}
```

```
if (q5Answered == false){
showResponse('incorrect5');
```

```

}
if (q6Answered == false){
showResponse('incorrect6');
}
if (q7Answered == false){
showResponse('incorrect7');
}

// Change the submitPressed variable to true to allow submission
to proceed next time and return 'false' to the form to prevent
submission this time.

submitPressed = true;
return false;
    }
}
</script>

</head>

<body>
<div class="form">
<form name="exampleform1" action="" method="post"
onSubmit="return checkForm(this);">
<p> 1. What is your name?

<!-- Name of the first layer. The other layers all have a unique
name which can be fed into the 'showResponse()' and
'hideResponse()' functions as arguments placed within the
brackets. -->

<span id="incorrect1" style="display:none">

<!-- Class to style the text in the layer (Enbolden and change
colour to red (See Style information in head of document) -->

<span class="response">*</span></span></p>

<!-- If adding further questions, add similar comments layers
next to each with a unique name. After adding validation for the
question to the JavaScript code, call the 'showResponse()' and
'hideResponse()' functions for this layer as appropriate in the
same way as is done for the 7 layers associated with the current
questions -->

<p><input type="text" name="namebox" /></p>
<p>2. How old are you? <span id="incorrect2"
style="display:none"><span class="response">* Please enter your
age as a number or leave the box blank</span></span></p>
<p><input name="agebox" type="text" size="5" maxlength="3" />
years</p>
<p>3. What is your gender (M or F)? <span id="incorrect3"
style="display:none"><span class="response">* Please enter your
gender as 'M', 'm', 'F' or 'f'</span></span></p>

```

```

<p><input name="genderbox" type="text" size="1" maxlength="1"
/></p>
<p>4. How often do you use the internet? <span id="incorrect4"
style="display:none"><span class="response">*</span></span></p>
<p>
<input type="radio" name="often" value="everyday" /> everyday<br
/>
<input type="radio" name="often" value="2-3 days per week" /> 2-3
days per week<br />
<input type="radio" name="often" value="4-5 days per week" /> 4-5
days per week<br />
<input type="radio" name="often" value="6-7 days per week" /> 6-7
days per week<br />
<input type="radio" name="often" value="less than once a week" />
less than once a week
</p>
<p>5. Which of the following do you regularly use the internet
for? <br />
(You can select as many options as you like. If you would like to
remove a selection you have made, select it again to deselect
it). <span id="incorrect5" style="display:none"><span
class="response">*</span></span></p>
<p>
<input type="Check box" name="uses" value="email" />E mail<br />
<input type="Check box" name="uses" value="finding information
about things to buy" />Finding information about things to buy<br
/>
<input type="Check box" name="uses" value="making purchases"
/>Making purchases<br />
<input type="Check box" name="uses" value="entertainment"
/>Entertainment<br />
<input type="Check box" name="uses" value="finding general
information" />Finding general information<br />
<input type="Check box" name="uses" value="educational courses"
/>Educational courses<br />
<input type="Check box" name="uses" value="downloading music"
/>Downloading music<br />
<input type="Check box" name="uses" value="discussion boards"
/>Discussion boards<br />
<input type="Check box" name="uses" value="real-time chat" />
Real-time chat</p>
<p>6. How would you rate your skill as an internet user? <span
id="incorrect6" style="display:none"><span
class="response">*</span></span></p>
<p>
<select name="skill">
<option selected="selected">Select an option</option>
<option>-----</option>
<option value="very advanced">Very advanced</option>
<option value="advanced">Advanced</option>
<option value="average">Average</option>

```



```

<option value="basic">Basic</option>
<option value="very basic">Very basic</option>
</select>
</p>
<p>7. What, in your opinion, are the three main advantages of the
internet?
<span id="incorrect7" style="display:none"><span
class="response">*</span></span></p>
<p><textarea name="advantages" rows="5" cols="50"></textarea>
</p>
<p><input type="submit" name="Submit" value="Submit" /></p>
<div id="incorrect" style="display:none">
<div class="response">
<p>Please check the answers highlighted with an asterisk (*). If
you are happy with them, press the 'submit' button again to
proceed with the submission.</p>
</div>
</div>
</form>
</div>
</body>
</html>

```

Validating before submission

The following form applies validation routines before the submit button is pressed. For both questions, participants are prevented from selecting more than three options, and for question 2, they are also limited to entering '1', '2', or '3' in the text boxes.

1. Choose THREE adjectives that you think best describe successful websites?
If you would like to change a selection you have made, select it again to deselect it.

- Easy to use
- Attractive
- Functional
- Fast
- Well-known
- Colourful
- Dynamic
- Interactive
- Simple
- Varied
- Up-to-date

2. From the following list of potential disadvantages of the internet, choose the THREE which you consider to be the most serious.

Then rank them by writing the numbers 1, 2 and 3 in the boxes. 1 = Most serious, 3 = least serious.

It is expensive	<input type="text"/>
It is too slow	<input type="text"/>
It provides information that is mainly of poor quality	<input type="text"/>
It offers too much information	<input type="text"/>
It wastes too much time	<input type="text"/>
It can expose the user's computer to virus attacks or hacking	<input type="text"/>
There are too many adverts and/or too much junk email	<input type="text"/>
There are often problems with broken links	<input type="text"/>

Three different validation routines are carried out.

The first two routines are carried out independently of the submit button when participants are completing the questions. The first occurs as the result of an **onclick** event handler, when participants select a Check box. The second occurs as a result of an **onblur** event handler when participants click away from a text box. (see the 'Event handlers' section of the 'Introduction to JavaScript' for more information if necessary).

This type of validation ensures that problems are dealt with before participants attempt to submit and also serves to provide on-going instructions to participants attempting to complete the questions.

Finally, there is a function called by pressing the submit button. This checks that three options have been selected for the questions and delivers an alert message if this is not the case.

Each of the routines is explained in the following sections:

'onclick' validation

The validation routine for question 1 consists of a function which is called by the **onclick** event handler when participants select the Check boxes.

Each of the eleven Check boxes is given the same name (**adjectives**) which means that they are automatically added to an array from zero to ten. They are also given appropriate values so that when the form is submitted the value of checked boxes can be added to the **adjectives** variable. The **onclick** event handler is added to each which calls a function called **count()** and returns the result of this function to the Check box. If **true** is returned, the Check box is selected, and if **false** is returned, it is deselected. (This works in a similar way to returning **true** or **false** to a form when it is submitted). Thus, the HTML for the first and second Check boxes is:

```
<input type="Check box" name="adjectives" value="easy"
onclick="return count(this.form)" />
Easy to use<br />
<input type="Check box" name="adjectives" value="attractive"
onclick="return count(this.form)" />
Attractive<br />
```

When participants select a box, the function counts the number of checked boxes, returning **false** (deselecting the box) if three boxes have already been selected, and returning **true** allowing the box to remain selected) if fewer than three have been selected. The commented code is as follows:

```
// start function and tell it to expect the name and reference to the form to be passed into it (to replace the word 'form') when the function is called.
function count(form) {
// Set variable called 'number' used to count the number of Check boxes selected. This is contained within the function, so it is 'local' to the function. Thus, it is 'reset' to 0 each time the function is called and does not 'remember' between function calls. A variable with the same name can be included in a different function without interfering with this one.
var number = 0;
// Start a loop to check each of the boxes. For each box, check if it is selected, and if so, add 1 to the 'number' variable.
for (i = 0; i < form.adjectives.length; i++) {
    if (form.adjectives[i].checked == true){
        number = number+1;
    }
}
// Check the 'number' variable. If it is less than 4, return 'true' so that it stays checked.
if (number < 4) {
    return true;
}
// If it equals 4 (the Check box clicked would be the fourth to be selected), return 'false' so that it is deselected.
else if (number ==4) {
    return false;
}
}
```

'onblur' validation

The routine for question 2 consists of a function called **check2()** which is called by the onblur event handler when participants click away from a text box. The counting of the text boxes that have been filled in occurs in a similar way to the routine in question 1.

However, the text boxes cannot be given the same name, as it would not be possible to distinguish which ones have been chosen. Thus they cannot be counted in an array. Instead of

this, it is possible to use the **form.elements** array to count the boxes. This is an array of all the form elements in a form. The first to appear is given the number zero (**form.elements[0]**) and the second is given the number one (**form.elements[1]**). This continues for all the elements in the form.

By counting how many elements come before the first text box in question 2, and counting how many text boxes there are, we can find the number of the boxes in this **form.elements** array and use this information in the validation routine. There are 11 Check boxes in question 1 (**form.elements[0]**) to (**form.elements[10]**) and there are 8 textboxes in question 2. Thus the first textbox in question 2 is the 12th in the **form.elements** array (**form.elements[11]**) and the last is the 20th (**form.elements[19]**).

This means that the routine needs to check the 12th to the 20th elements and count the ones that have been filled, preventing more than three from being filled. This is similar to the code in the routine for question 1. However, the number of the box in the **element.array** must be fed into the function as an argument so that the box that called the function can be identified.

Thus, there are two arguments in the function (the form reference and the number of the box in the array), and the HTML for the first two boxes is as follows:

```
<p> It is expensive  
<input type="text" maxlength="1" size="1" name="expensive"  
onblur="count2(this.form, 11)" /><br />  
It is too slow  
<input onblur="count2(this.form, 12)" type="text" maxlength="1"  
size="1" name="slow" />
```

and the function is begun and ended as follows:

```
function count2(form, box) {  
}
```

The section of the function that counts the boxes is as follows:

```
// Set variable called 'number' used to count the number of  
textboxes filled.  
  
var number = 0;  
  
// Start a loop to check each of the boxes. For each box, check  
if it is filled (its value is not an empty string(""), and if so,  
add 1 to the 'number' variable.  
  
for (i = 11; i < 19; i++) {  
    if (form.elements[i].value != ""){  
        number = number+1;  
  
// Check the 'number' variable. If it is has reached 4 (this is  
the fourth box to be filled), delete the contents of the last box  
to be filled in (by changing the value of the box which called  
the function to "") and provide an alert message.  
  
        if (number == 4){
```

```

        form.elements[box].value = "";
        alert("Please select three options only. Delete one of your
        selections if you would like to change it.");
    }
}
}

```

// NB. It is important to check that all curly brackets that have been opened are closed correctly, or the script will not run correctly.

This provides the facility of limiting the number of responses to three, but it does not limit what the contents of the boxes can be.

To do this, it is necessary to check that only the numbers 1, 2 and 3 have been entered in the boxes, and to ensure that these numbers have been entered only once.

Three variables are set to count the number of times that 1, 2 and 3 have been entered and if a box that calls the function is found to contain a number for the second time, the contents are deleted and a message delivered to the participant. This also occurs if any other content is inserted into the box.

The commented code for this section of the function is as follows:

// Set variables called 'onePicked', 'onePicked', and 'onePicked' used to count the number of occurrences of 1, 2 and 3.

```

var onePicked = 0;
var twoPicked = 0;
var threePicked = 0;

```

// Start a loop to check each of the boxes.

```

for (j = 11; j < 19; j++) {

```

// For each box, check if the value is "1", and if so, add 1 to the 'onePicked' variable. If it has reached 2 ('onePicked is greater than 1), delete the contents of the last box to be filled in (by changing the value of the box which called the function to "") and provide an alert message.

```

if (form.elements[j].value == "1") {
    onePicked = onePicked +1;
    if (onePicked >1){
        form.elements[box].value = "";
        alert("Please select rank 1 for one option only");
    }
}
}

```

// Do the same for value "2".

```

else if (form.elements[j].value == "2") {

```

```

twoPicked = twoPicked +1;
if (twoPicked >1){
    form.elements[box].value ="";
    alert("Please select rank 2 for one option only");
}
}

// And for value "3".

else if (form.elements[j].value == "3") {

    threePicked = threePicked +1;
    if (threePicked >1){
        form.elements[box].value ="";
        alert("Please select rank 3 for one option only");
    }
}

// Finally check that no other value has been added to the boxes,
// by checking for boxes that are not empty (and do not contain "1",
// "2" or "3". If any other value is found, delete the contents of
// the box, and give focus to it (place the cursor in the box) ready
// for the participant to add a different value.

else if (form.elements[j].value != ""){

    alert("Please enter a number from 1-3");
    form.elements[box].value ="";
    form.elements[box].focus();
}

}

```

'onSubmit' validation

This is the most straightforward of the functions, because the more involved validation activities have been carried out beforehand. All that has to be done is to check that three options have been chosen for each question. If this is the case, it can be assumed that the validation from the other functions has already ensured the correct number and format of responses has been provided.

The commented code is as follows:

```

function checkForm(form) {

// Set variable used to count the number of Check boxes selected
// for question 1.

var number = 0;

// Start a loop to check each of the boxes in question 1. For
// each box, check if it is checked, and if so, add 1 to the
// variable.

for (i = 0; i < form.adjectives.length; i++) {

```

```

        if (form.adjectives[i].checked == true){
            number = number+1;
        }
    }

    // At the end of the loop, check how many boxes have been
    // selected. If it is not three, deliver an alert message and return
    // 'false' to the form to prevent submission.

    if (number != 3) {
        alert("Please select three options for question 1.");
        return false;
    }

    // Set variable used to count the number of text boxes filled in
    // for question 2.

    var number2 = 0;

    // Start a loop to check each of the boxes in question 2. For
    // each box, check if it is filled (it's value is not an empty
    // string ""), and if so, add 1 to the variable.

    for (j = 11; j < 19; j++) {
        if (form.elements[j].value != "") {
            number2 = number2+1;
        }
    }

    // At the end of the loop, check how many boxes have been filled.
    // If it is not three, deliver an alert message and return 'false'
    // to the form to prevent submission.

    if (number2 != 3) {
        alert("Please select your three options for question 2.");
        return false;
    }

    // If the code makes it to this stage without returning 'false'
    // to the form, the correct number of responses have been made.
    // Deliver a thank-you alert message and return 'true' to the form
    // to allow submission to proceed.

    alert("Thank you for completing the form.");
    return true;
}

```

The complete HTML and code is as follows:

```

<html>
<head>
<title>Exploring ORMs | Example of validating multiple form
elements</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-
8859-1">
<link href="../generic/nn4.css" rel="stylesheet" type="text/css">
<style type="text/css">
@import url(../generic/main.css);
</style>
<script language="javascript" type="text/javascript">

function count(form) {
var number = 0

for (i = 0; i < form.adjectives.length; i++) {
  if (form.adjectives[i].checked == true){
    number = number + 1
  }
}

  if (number < 4) {
return true;
}

  else if (number == 4) {
return false;
}
}

function count2(form, box) {

var onePicked = 0;
var twoPicked = 0;
var threePicked = 0;
var number = 0;

for (i = 11; i < 19; i++) {
if (form.elements[i].value != ""){
  number = number + 1;
  if (number == 4){
    form.elements[box].value = "";
    alert("Please select three options only. Delete one of your
selections if you would like to change it.");
  }
}
}

}

for (j = 11; j < 19; j++) {
  if (form.elements[j].value == "1") {
onePicked = onePicked +1;
  if (onePicked >1){

```



```

        alert("Please select rank 1 for one option only");
        form.elements[box].value = "";
    }
}
else if (form.elements[j].value == "2") {
    twoPicked = twoPicked + 1;
    if (twoPicked > 1){
        alert("Please select rank 2 for one option only");
        form.elements[box].value = "";
    }
}
else if (form.elements[j].value == "3") {
    threePicked = threePicked + 1;
    if (threePicked > 1){
        alert("Please select rank 3 for one option only");
        form.elements[box].value = "";
    }
}
else if (form.elements[j].value != ""){
    alert("Please enter a number from 1-3");
    form.elements[box].value = "";
    form.elements[box].focus();
}
}
}

function checkForm(form) {
    var number = 0;
    for (i = 0; i < form.adjectives.length; i++) {
        if (form.adjectives[i].checked == true){
            number = number+1;
        }
    }
    if (number != 3){
        alert("Please select three options for question 1.");
        return false;
    }
    var number2 = 0
    for (j = 11; j < 19; j++) {
        if (form.elements[j].value != ""){
            number2 = number2+1;
        }
    }
    if (number2 != 3){
        alert("Please select your three options for question 2.");
        return false;
    }
}

```

```

alert("Thank you for completing the form.");
return true;
}
</script>

</head>

<body>

<form name="check" action="" method="post" onSubmit="return
checkForm(this);">
<div class="ques">
<p>1. Choose THREE adjectives that you think best describe
successful websites?
<br>
If you would like to change a selection you have made, select it
again
to deselect it. </p>
<p>
<input type="Check box" name="adjectives" value="easy"
onclick="return count(this.form)" />
Easy to use<br />
<input type="Check box" name="adjectives" value="attractive"
onclick="return count(this.form)" />
Attractive<br />
<input type="Check box" name="adjectives" value="functional"
onclick="return count(this.form)" />
Functional<br />
<input type="Check box" name="adjectives" value="fast"
onclick="return count(this.form)" />
Fast<br />
<input type="Check box" name="adjectives" value="well-known"
onclick="return count(this.form)" />
Well-known<br />
<input type="Check box" name="adjectives" value="colourful"
onclick="return count(this.form)" />
Colourful<br />
<input type="Check box" name="adjectives" value="dynamic"
onclick="return count(this.form)" />
Dynamic<br />
<input type="Check box" name="adjectives" value="interactive"
onclick="return count(this.form)" />
Interactive<br />
<input type="Check box" name="adjectives" value="simple"
onclick="return count(this.form)" />
Simple<br />
<input type="Check box" name="adjectives" value="varied"
onclick="return count(this.form)" />
Varied<br />
<input type="Check box" name="adjectives" value="up-to-date"
onclick="return count(this.form)" />
Up-to-date </p>

```

<p>2. From the following list of potential disadvantages of the internet, choose the THREE which you consider to be the most serious.
 Then rank them by writing the numbers 1, 2 and 3 in the boxes. 1 = Most serious, 3 = least serious.

</p>

<p> It is expensive

<input type="text" maxlength="1" size="1" name="expensive" onblur="count2(this.form, 11)" />

It is too slow

<input onblur="count2(this.form, 12)" type="text" maxlength="1" size="1" name="slow" />

It provides information that is mainly of poor quality

<input onblur="count2(this.form, 13)" type="text" maxlength="1" size="1" name="poor info" />

It offers too much information

<input onblur="count2(this.form, 14)" type="text" maxlength="1" size="1" name="wastes time" />

It wastes too much time

<input onblur="count2(this.form, 15)" type="text" maxlength="1" size="1" name="viruses or hacking" />

It can expose the user's computer to virus attacks or hacking

<input type="text" maxlength="1" size="1" name="advertises and junk mail" onblur="count2(this.form, 16)" />

There are too many adverts and/or too much junk email

<input onblur="count2(this.form, 17)" type="text" maxlength="1" size="1" name="broken links" />

There are often problems with broken links

<input onblur="count2(this.form, 18)" type="text" maxlength="1" size="1" name="textfield9" />

</p>

<p>

<input type="submit" name="Submit" value="Submit" />

</p>

</div></form>

</body>

</html>

Accessibility

Providing that submission of a questionnaire is not prevented in browsers without JavaScript, validation routines will not prevent a questionnaire from being accessible as they will simply not work. Though this may have an impact on the quality of the data received from a non-JavaScript browser (e.g. if participants using these browsers accidentally miss questions or answer them in an inappropriate format), it will not prevent the questionnaire from being fully usable by these participants.

However, it may be beneficial to perform server-side validation as a 'back up' to the client-side JavaScript validation. Because server-side processing occurs before the HTML page is delivered to the browser, it is not dependent on the technology available on the client computer. Although this is a slower method of checking the information, it is likely to be necessary only where JavaScript is not available and thus where any problems with the data have not been dealt with before the form is sent. This kind of 'back-up' validation also has security benefits where users may have deactivated JavaScript in the browser to avoid validation for any malicious reasons.

Server-side validation

Methods of validating in any of the main server-side languages are basically similar to those using JavaScript, involving conditionals to check the contents or selection of form elements and carrying out different actions accordingly (e.g. processing results by emailing them or by adding them to a database, or preventing processing and returning a message to the participant prompting them to check and resubmit).

An overview of server-side technologies and their use in gathering data via questionnaires can be found in the 'Server-side processing 1: Introduction & E-mailing results' sections of this guide.

Depending on the technology used, the following links may provide a useful source of information, tutorials and examples on how to add server-side validation.

PHP/MySQL

A relatively straightforward introduction to validation using PHP.

http://webmonkey.wired.com/webmonkey/99/21/index4a_page2.html?tw=programming

A PHP tutorial which covers a range of different validation activities, leading to an example of a complete form validated via PHP.

<http://codewalkers.com/tutorials/47/2.html>

Sklar, C. (2004) *Learning PHP*. Sebastapol, CA: O'Reilly.

Chapter 6: Making web forms.

<http://www.oreilly.com/catalog/learnphp5/>

Coggeshall, J. (2005) *PHP Unleashed*. Indianapolis: SAMS.

Chapter 4: Working with Forms in PHP.

Chapter 5: Advanced Form Techniques.

<http://www.sampublishing.com/title/067232511X>

ASP.NET

Microsoft's ASP.NET framework offers a number of ready-made web controls designed to carry out server-side and, where available, client-side validation of web forms. These include controls that check required fields have been completed, that check that data in particular

ranges or patterns has been entered (e.g. in the format of a telephone or credit-card number), and that compare data from one form element for consistency with that from another.

A useful tutorial on how these controls work with code examples is available from the ASP.NET Quickstart tutorials

<http://beta.asp.net/QuickStartv20/aspnet/doc/validation/default.aspx>

A step-by-step guide is also available from the 4guysfromrolla.com site:

<http://www.4guysfromrolla.com/webtech/090200-1.shtml>

Mitchell, S. (2003) *Teach yourself ASP.NET*. Indianapolis: SAMS.

Chapter 12. Validating User Input with Validation Controls.

<http://www.sampublishing.com/title/0672325438#>

Walther, S. (2003) *ASP.NET. Unleashed*. Indianapolis: SAMS.

Chapter 2: Building Forms with Web Server Controls.

Chapter 3: Performing Form Validation with Validation Controls.

<http://www.sampublishing.com/title/0672325438#>

PERL/CGI

An introductory tutorial for adding validation using PERL.

http://www.elated.com/tutorials/programming/perl/cgi/form_validation/

Guelich, S., Gundavaram, S. and Birznieks, G. (2000) *CGI Programming with Perl*.

Sebastapol, CA: O'Reilly.

Chapter 4: Forms and CGI;

Chapter 8: Security (available as a sample chapter).

<http://www.oreilly.com/catalog/cgi2/toc.html>

Key design issues

Introduction

This section aims to provide the necessary skills to ensure that good design practices are followed in the creation of an online questionnaire. Alongside other aspects, it provides the supporting technical information for the design practices outlined in the design section of the online questionnaires module. The following information is covered:

- Deciding whether to present a questionnaire as a one-page web form or as one that spans multiple pages;
- Implementing a progress bar in multiple-page questionnaires;
- Delivering instructions via pop-up windows and alert boxes;
- Ensuring accessibility;
- Ensuring consistency: Browser issues, screen size, and the use of colour and font.

Single or multiple-paged questionnaires

One-page questionnaires are generally easier to implement as they consist of a single web form which can easily be processed through the submission of this one form. Where the questionnaire is relatively short and straightforward in terms of structure, this is likely to be the best option.

However, with longer or more complex forms, attempting to present the entire questionnaire in one page may lead to problems such as the following:

1. Presenting all questions at the same time may give an impression of greater length which may discourage participants from proceeding.
2. Opportunities to validate individual questions or smaller groups of questions as the participant progresses through the questionnaire may be reduced (see the 'Form validation' section). In turn, this may lead to frustration if all questions are validated at once at the end of the questionnaire.
3. Although skip patterns can be introduced through linking to anchors further down a page (see 'links' section of the 'Introduction to HTML 2' page) or through instructing participants to skip a question by scrolling to the next, this may not be the most effective or intuitive method of delivering the questions.
4. If a participant drops out mid-questionnaire, all data will be lost and there will be no opportunity for collection of partially-completed questionnaires or for identification of questions that may be precipitating drop out.

For longer questionnaires, the use of multiple pages can add to the effectiveness of question delivery, providing clearer routes through the questions and offering the opportunity for a more sophisticated presentation of skip patterns. For example, links to different sections can be added which participants can be prompted to select according to the answer to key questions. However, because all the questions are not made visible, submitted and processed at the same time, a number of extra aspects must be considered

1. An indication of progress through the questionnaire must be given, either through the use of a progress bar (see below) or through structuring the questionnaire into different sections and indicating the nature of this structure to respondents (see the 'Design of online questionnaires 1: Appearance' section of the questionnaires modules). If this is not done effectively, uncertainty over



(The width of the first table cell has been increased to 60%).

Similar effects could also be achieved through the use of CSS to style the table and cells.

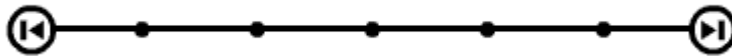
If this method is chosen, it is important to be aware that there may be some unpredictability in the way that the table cells render in different browsers. Thus it is important to test in different browsers and to consider supporting the visual representation of progress with text.

Graphics

An alternative to this is to create a series of graphics using a graphics program such as Jasc Paint Shop Pro, Adobe Photoshop, or Macromedia Fireworks. These can then be added at the top or bottom of each page. This has the advantage of allowing the creation of a more sophisticated or professional-looking progress bar, but it depends on having some familiarity with how these programs work (although it is relatively easy to create basic images using auto shapes such as circles or rectangles). Depending on the size of the images used, it may also have an impact on the overall file size of the questionnaire, possibly increasing download times and leading to increased non-response. If these images are used, alt text should also be added to provide an alternative description for users of text-only browsers.

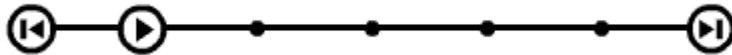
e.g.

Page 1

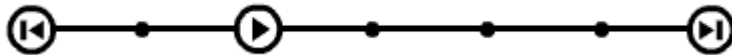


```
<p></p>
```

Page 2



Page 3



Delivering instructions via pop-up windows and alert boxes

When offering specific instructions for different question types, it can be useful to include optional instructions which can be accessed by the user when required or delivered where a user experiences difficulty. This can be achieved through the use of simple JavaScripts to insert instructions in alert boxes and pop-up windows.

Alert boxes

The following is an example of a link which displays an alert box message when clicked:

[Click to see the alert box message](#)

The code to produce this link is as follows:

```
<a href="JavaScript:void(0);" onclick="alert('Alert box message is added here...');">Click to see the alert box message</a>
```

The link `JavaScript:void(0);` is an 'empty' link so that when the user clicks on it, a new document is not loaded, but some JavaScript is carried out. The action `alert('Alert box message is added here...')` is triggered by the `onclick` event handler when the user clicks on the link. The contents of the alert box are placed within inverted commas. It is important to ensure that the message does not include inverted commas as this would signal the end of the message and lead to an error which would prevent the box being displayed. Rather than using the inverted comma mark (`'`), the special character codes `‘` (`'`) and `’` (`'`) should be used to prevent this.

e.g.

[Click to see the use of special characters rather than inverted commas](#)

is produced by:

```
<a href="JavaScript:void(0);" onclick="alert('It&#8217;s important not to use inverted commas. Special characters should be used instead (&#8216; and &#8217;);')">Click to see the use of special characters rather than inverted commas.</a>
```

It is also possible to change the message in the status bar at the bottom of the browser window to provide instructions on how to use the link when the user places his/her mouse over the link. In the following link, the status bar text is changed to 'Click to see the alert box message'.

[Hold your mouse over the link to see the changed status bar text.](#)

```
<a href="JavaScript:void(0);" onclick="alert('Alert box message added here...');" onmouseover="window.status = 'Click to see the alert box message'; return true;" onmouseout="window.status = '';">Hold your mouse over the link to see the changed status bar text.</a>
```

The code for changing the text is `window.status = "Add status bar text here..."`. This is triggered by an `onmouseover` event handler so that the action occurs when the mouse is placed over the link.

The action triggered by the `onmouseout` event when the user's mouse moves away from the link deletes the text in the status bar, replacing it with an empty string (`""`).

In both cases, it is necessary to add the syntax `return true;` to preserve the text (or empty string) that has been set until another instruction to change the status bar text is reached.

To ensure that the instructions remain accessible when a user accesses them using the keyboard rather than a mouse, it is also important to add the `onfocus="this.onmouseover();"` and the `onblur="this.onmouseout();"` code, which ensures that the same actions will take place when a user without a mouse tabs to or away from the image.

```
<a href="JavaScript:void(0);" onclick="alert('Alert box message added here...');" onmouseover = "window.status = 'Click to see the alert box message'; return true;" onmouseout = "window.status = '';" onfocus="this.onmouseover();" onblur="this.onmouseout();" >Tab to the link or hold your mouse over it to see the changed status bar text.</a>/>
```

Pop-up windows

In a similar way, JavaScript can be used to produce a link which opens a pop-up window when clicked. To create such a link, the following JavaScript is placed in the head of the document:

```
<script language="javascript" type="text/javascript">

function popInstr(msg){
var win = window.open ('blank.htm', '',
'width=300,height=75,left=150,top=150scrollbars=yes');
win.document.write("<html><head><title>Example of 'pop-up' instructions</title></head><body><p>" + msg + "</p><p><a href='JavaScript:void(0);' onclick='window.close()'>CLOSE </a></p></body></html>");
win.document.close();
}

</script>
```

The following HTML/JavaScript is then placed in the body of the document:

```
<p>Select the following link to <a href="JavaScript:void(0);" onclick="popInstr('Further instructions in a pop-up window');" onmouseover = "window.status = 'Click to see the pop-up window'; return true;" onmouseout = "window.status = '';">further instructions</a> if required. These will open in a pop-up window so you will need to disable any pop-up blocking software to see them.</p>
```

Learning activity

Examine the different parts of the code to try to establish what their functions are. Refer to the explanation below to check.

```
<script language="javascript" type="text/javascript">

function popInstr(msg){
var win = window.open ('blank.htm', '',
'width=300,height=75,left=150,top=150scrollbars=yes');
win.document.write("<html><head><title>Example of 'pop-up'
instructions'</title></head><body><p>" + msg + "</p><p><a
href='JavaScript:void(0);' onclick='window.close()'>CLOSE
</a></p></body></html>");
win.document.close();
}

</script>

<a href="JavaScript:void(0);" onclick="popInstr('Further
instructions in a pop-up window');" onmouseover = "window.status =
'Click to see the pop-up window'; return true;" onmouseout
="window.status = '";>
```

Explanation

Section of code	Explanation
<code><script language="javascript" type="text/javascript"></code>	Scripts in a document must be surrounded by the script tags <code><script></script></code> . This tells the browser that what follows is a script and indicates which scripting language is being used to allow it to interpret the information correctly.
<code>function popInstr(msg){</code>	This declares a function called <code>popInstr</code> and tells it to expect an argument called <code>msg</code> to be included when the function is called. See the 'Introduction to JavaScript' section for more information about functions and arguments if required. The function is surrounded by curly brackets (<code>{}</code>).
<code>var win = window.open ('blank.htm', 'blank', 'width=300,height=75, left=150,top=150,scrollbars=yes');</code>	This declares a variable called <code>win</code> which consists of a new window created using the <code>window.open</code> method. The new window opens an HTML document saved in the same folder called <code>blank.htm</code> which is given the name <code>blank</code> . This window is set to a size of 300x75 pixels and is placed 150 pixels from the top-left corner of the user's screen. It is given scrollbars, but no tool bar, status bar or menu bar. It is important to add all the settings information with no spaces to ensure that the browser can recognise it correctly.
<code>win.document.write("<html><head><title>Example of 'pop-up' instructions'</title></head><body><p>" + msg + "</p><p>CLOSE </p></body></html>");</code>	This writes HTML to the new window (<code>win</code>) using the <code>document.write</code> method. Everything within the brackets is written to the document in the window. The contents are added as strings between inverted commas. The syntax <code>' + msg + '</code> tells the function that the argument called <code>msg</code> should be fed in to the function at this point when it is called. The plus

	<p>signs are used to add this argument to the contents of the two strings on either side (i.e. the message is added to the HTML written to the new window). See the 'Introduction to JavaScript' section for more information about arguments if necessary.</p> <p>The HTML written to the window includes an 'empty' link with an onclick event handler that closes the window when the link is clicked using the <code>window.close()</code> method.</p> <pre>(CLOSE)</pre> <p>CSS or HTML attributes and values could be added at the relevant points to change the style of the contents of the new document.</p>
<pre>win.document.close();</pre>	<p>The <code>document.close</code> method is then added to indicate that the process of writing to the window (<code>win</code>) is complete.</p>
<pre></pre>	<p>This is an 'empty' link which means that when the user clicks on it, a new document is not loaded, but a JavaScript function is carried out. There are three event handlers <code>onclick</code>, <code>onmouseover</code>, and <code>onmouseout</code>. When the user select the link (<code>onclick</code>), the function <code>popInstr</code> is called and the argument <code>'Further instructions in a pop-up window'</code> is fed in to the function. When the user holds his/her mouse over the link (<code>onmouseover</code>), the status bar text changes to <code>'Click to see the pop-up window'</code>. When the user moves his/her mouse away from the link (<code>onmouseout</code>), the status bar text is deleted, changing to an empty string (<code>'</code>).</p>

For further information see the following link for a range of JavaScript examples with source code specifically related to windows.

<http://www.irt.org/script/window.htm>

N.B. It should be remembered that some users may not have JavaScript activated in the browser and the use of alternative instructions should be included which will display for users without JavaScript through the `<noscript>` tag - see 'Introduction to JavaScript' section.

Accessibility

There are a number of simple steps that can be taken to increase the accessibility of an online questionnaire and its associated web pages. These can ensure that the contents are accessible to users with a range of user-agents including text-only and screen reading browsers and other assistive technologies, and mobile web-enabled devices.

Designing the site to be compliant with standards set out by the World Wide Web Consortium (W3C) is an important step in ensuring accessibility. The resources section of this guide includes a link to W3C's validation tools which allow web pages to be checked for standards compliance. By uploading a page or entering the URL, the tools will run automated tests and report on any pieces of invalid markup in the pages.

It is also good practice to separate content from presentation in web pages by using Cascading Style Sheets (CSS) to add design features (See the 'Introduction to CSS' section of this guide). This allows the user to control how the site should be presented, allowing style information to be overridden to allow presentational features such as text size, font, colour and layout to be changed according to need or preference. It also maximises the possibilities that the site will

be fully accessible to a range of user agents (e.g. browsers, media players and assistive technologies).

Beyond this, the W3C Web Accessibility Initiative guidelines (WAI) includes a wide range of measures which should be taken to ensure that a website is accessible (<http://www.w3.org/WAI/intro/wcag.php>). The guidelines divide these measures into Priority 1, 2 or 3 according to how essential they are to accessibility. Some key measures that should be taken for accessibility are shown below.

Key measures

Some key measures that should be taken to ensure accessibility are:

- If scripting and third-party plug-ins (e.g. Flash) are used, ensure that all content remains accessible when these are not available in the user's browser. Provide alternatives to scripts if necessary through the use of **<noscript></noscript>** tags (see the 'Introduction to JavaScript' section of this guide) and provide text-only alternatives to multimedia;
- Make explicit links between form labels and form elements using the **<label for="name of element">** tag, and make sure that there is a logical tab order within forms to make them fully useable and to ease navigation via keyboard (see the 'Web forms' section);
- Provide text alternatives for graphics to ensure the site contents can be accessed in non-graphical browsers. It is important to make sure that this provides an adequate description of the value of the image on the page. Descriptions are added using the **alt="Add description"** attribute of the image tag as follows:

```

```

- Use descriptive links to ensure links make sense out of context and make them accessible to user agents that present all links on a page as a simple list. e.g.

Select the following link for **[more information about accessibility](#)**

rather than

For more information about accessibility, **[click here](#)**

- Ensure that colour schemes are appropriate so that there is adequate contrast between the colours used. Ensure that different colours are not used to provide information that can not be otherwise gleaned by users who can not distinguish between them
- A service called 'Vischeck' allows pages to be tested for suitability for colour-blind users. The service simulates the appearance of pages to users with different forms of colour blindness:
<http://www.vischeck.com/vischeck/>;
- Avoid excess movement on screen or flickering screens caused by elements such as animations or 'blinking text' which can cause problems for epileptic users.
- Use relative sizing to make contents flow into viewing areas of different sizes and to allow users to control the size of text through their browser settings. e.g.:

`<table width="95%">`

rather than

`<table width="560">`

- Use the `summary="Add description"` attribute to describe the purpose of tables, especially if they are used for layout rather than data. Organise data in tables to ensure that it makes sense if read by screen-reading software (usually row by row from left to right).
- Use consistent and clear navigation, colours and icons and make language clear and as simple as possible for the purpose and intended audience of the questionnaire. This will make the contents as accessible as possible to users who may have reading and/or cognitive disabilities.

Consistency

The following issues need to be considered alongside the accessibility information given above. This will ensure that the questionnaire appears as consistently as possible in different browsers and on different computer systems.

Browser issues

It is important to design a questionnaire with an awareness that participants may be using a range of different browsers and that the way the questionnaire looks in these browsers may vary.

Information on the main browsers available at the time of writing is available on the Worldwide Web Consortium (W3C) website at the link below along with statistics on the usage levels of each (though it should be noted that the statistics are based on users of the site and should thus be generally considered to be skewed in favour of more technically proficient users).

<http://www.w3schools.com/browsers/default.asp>

At the time of writing, estimates suggest that at least three-quarters of web users have Microsoft Internet Explorer (many suggest that the proportion is substantially higher than this), with the vast majority of the remainder using the Mozilla Firefox browser.

Maintaining a straightforward design for the questionnaire, ensuring it is designed for accessibility (see above) and validating the HTML and or CSS used to create it (see the 'Resources' section of this guide for further information if required) will limit differences of appearance in different browsers. However differences in aspects such as the size and appearance of form elements and tables may remain. It is thus good practice to test pages on as many different browsers and systems as possible, and as an absolute minimum to install the latest versions of the three most popular browsers on your desktop and use these to test. Friends and acquaintances with different systems (e.g. AppleMacs or PCs) and older versions of browsers can also be called on to test for any problems.

Where design problems are found in particular systems and browsers, an attempt can be made to change the design to best accommodate them, bearing in mind usage statistics, the context of the research, and the overall balance of probability of a participant having this browser.

It is also possible to use browser detection JavaScript to identify which browser is being used. A number of tutorials on how to do this as well as free scripts can be found using a simple search for 'browser detection' or 'browser sniffing'. The browser information can then be used to deliver different designs tailored to particular browsers. This can be done either by linking

to different CSS files designed for different browsers, or redirecting to different versions of the same questionnaire. However, when considering this, it should be remembered that users may not have JavaScript activated in the browser - see the 'Introduction to JavaScript' section). Aiming for an accessible and relatively straightforward design should avoid the necessity for browser detection.

It is also possible to collect information about the user's computer and browser alongside the data from the questionnaire (see the 'Gathering information about participants' section of this technical guide. This can allow an overview of the technologies available to respondents to be gained. If it becomes clear, for example, that older or less common browsers are being used to access the questionnaire, it can be tested on these browsers and redesigned if necessary.

Plug-in detection

Where a questionnaire includes multi-media, it should be remembered that users may not have the necessary technology installed. In many cases, the user's browser will automatically detect that a particular plug-in is required and prompt the user accordingly, although many users may be reluctant to proceed with installing software on their machines. Clearly specifying minimum specifications required for the questionnaire and adding an easily accessible link to any plug-in software required may increase the chances of the user doing this.

JavaScript browser detection (or 'browser sniffing') can also be used to test whether or not the user has certain technology installed. If, for example, Macromedia Flash content has been used extensively to deliver video or animation as part of the questionnaire, it may be necessary to redirect the user to instructions or alternative content if he or she does not have a suitable version of Flash installed. The following web site provides a good explanation of Flash detection (and its limitations) along with free scripts to carry it out:

<http://moock.org/webdesign/flash/detection/moockfpi/>

The latest version of Macromedia Flash also allows for files to be published with automatic detection of which version of the Flash player is being used to access them, followed by automatic redirection to an alternative site where the version is not adequately up to date.

Screen size

When designing a questionnaire, it is important to try to ensure that it can be viewed successfully on a range of different screen sizes. At the time of writing, the most common screen size is 800 x 600 pixels (with an approximate typical page viewing area of 745 x 415 pixels when the space taken by the browser toolbars is considered). However other common screen sizes are:

Screen size	Typical viewing area
640 x 400 pixels	585 x 295 pixels
1024 x 768	970 x 580
1280 x 1024	1225 x 840

Increasingly, web statistics such as those from the Worldwide Web Consortium (W3C) indicate that very few computers have screen settings of 640 x 400, and it is common practice for web developers to design their pages with 800 x 600 as a notional minimum.

If, however, a researcher wishes to ensure that his or her questionnaire will be fully accessible to those who may be using older equipment, he or she may decide to design for a 600 x 640 minimum. This assumes that any items such as images or tables which are larger than 585 x 295 pixels may not be fully visible in the browser window that and scrolling will be required.

Limiting width is particularly important as horizontal scrolling tends to produce serious usability problems which may affect response rates and validity of results. Although vertical scrolling does not tend to present such serious usability problems in general web use, this may

need to be given particular consideration for online questionnaires. For example, if the possible responses to a particular question are not all visible at the same time, this may have an effect on results. For this reason, keeping the height of each question to a maximum of 295 pixels should be considered.

To test how a questionnaire appears at different sizes, a link to a pop-up window set to the test size can be used. The HTML / JavaScript to achieve this is as follows:

```
<p><a href="JavaScript:void(0);"
onclick="window.open('question.htm','test','width=585,height=295,
scrollbars=yes')">Screen size test</a></p>
```

It is necessary simply to replace 'question.htm' with the path to the test page, and to change the width and height to the desired test size.

Colour

When choosing a colour scheme for the questionnaire it is recommended that web-safe colours are used to ensure consistency. The 'web-safe' colour palette consists of 216 colours which are not subject to variation on different types of monitors and systems. The following link provides a palette of web-safe colours organised by either hue (colour) or value (lightness). This makes it easier to design appropriate colour schemes using these colours.

<http://www.lynda.com/hex.html>

Font

When choosing a font for use in the questionnaire, it is important to remember that the user's computer may not have particular fonts that you may wish to use. If the user's browser automatically replaces a font that you have used, this may have an impact on the layout and usability of your questionnaire, e.g. by having unanticipated effects on the spacing of possible responses or increasing the width of items beyond the smallest screen sizes.

Using common fonts is recommended, as is the use of 'font-families' which provide the browser with information about which fonts should be used as a replacement in a case where a particular font is not available. Thus the use of the HTML tag **** (or the CSS rule **body { font-family: Verdana, Arial, Helvetica, sans-serif}**) tells the browser to use Arial if Verdana is not available, followed by Helvetica, and finally by the default sans-serif font. The questionnaire can then be checked with each of these fonts to ensure that the size, spacing and layout is suitable with each.

Gathering information about participants

Introduction

Beyond the survey data, there are a number of possibilities for the collection of information that may be of use to the researcher.

This page will describe how information can be stored in hidden form fields to be submitted alongside the data from visible form fields.

It will also provide an overview of how JavaScript and a technology called 'server-side includes' can be used to collect the following:

- IP addresses and date and time information including approximate completion time which may be useful for security and identifying and removing anomalous submissions.
- The URL of the referring page (the page which contained the link the participant followed to reach the questionnaire) which may be useful in providing information about the success of the forms of advertising used to elicit participation.
- Information about the user's computer and browser such as whether JavaScript and cookies were enabled, the screen-size of the monitor used, or which type and version of browser was used to access the questionnaire. This information may prove valuable in determining whether some features such as validation were available when the participant completed the questionnaire. It is also likely to be useful in gaining an overview of the technologies available to respondents. If it becomes clear, for example, that older or less common browsers are being used to access the questionnaire, it can be tested on these browsers and redesigned if necessary.

Finally, it will outline how cookies can be saved onto a participant's computer so that action can be taken if the same computer is used to access the questionnaire for a second time.

Adding data to hidden form fields

As their name suggests, hidden form fields and their contents are not displayed in the browser. They can thus be used to submit information along with the form data which it is not necessary or desirable to display. It should be remembered, however, that a participant can access the information in a hidden form field simply by opting to view the HTML source code for the page.

Adding information to a form in a hidden form field is straightforward. The code is as follows:

```
<input type="hidden" name="hidden1" value="value added by the researcher or set via HTML or JavaScript" />
```

When the form is submitted, the contents of the form field (the value) are added alongside the data from visible form fields. Thus, for example, if more than one version of a questionnaire is released, the version number can be added to the form as follows:

```
<input type="hidden" name="version" value="1" />
```

When data is submitted, it would thus be clear which version of the questionnaire was completed.

Client-side or Server-side information gathering

Where the ability to gather user information is desired as an 'extra' in cases where the researcher is able to accept that it may not be possible to gather information from all participants, the use of client-side information gathering using JavaScript is ideal as it allows a straightforward way of collecting a wide range of information. It does not matter what type of server the questionnaire is hosted on as the processing that collects the information and adds it to the form data is carried out on the user's computer.

In cases where the gathering of user information is essential to a study, however, it is recommended that the process is carried out using server-side processing which will operate regardless of the technologies available on the client computer. This is because JavaScript may not be available to all browsers and because users may choose to disable JavaScript on their machines.

A further advantage of using server-side processing is that it is more likely to allow accuracy and consistency of information to be achieved. For example, JavaScript relies on the accuracy of the time and date on the user's computer which may be open to variation across different machines. Server-side technologies, however, take this from a single source (the server). It is relatively straightforward to gather a range of information using ASP.NET or PHP, depending on the server technology used. (see the 'Server-side processing' sections for more information on these technologies).

For researchers with a server capable of supporting them, Server-Side Includes (SSI) provide perhaps the most straightforward and easily-implemented way of gathering user information through server-side processes. This page will focus on the use of this technology alongside JavaScript.

Data gathering via Server-Side includes (SSI)

Server-Side Includes are designed to allow small pieces of a web page to be dynamically generated. They are frequently used in web pages to add information such as the date the page was last modified or the current time and date. SSI pages are generally marked with an **.shtml** suffix which signals to the server that any SSI directives should be processed server-side.

Where the questionnaire is hosted on a server that supports them, they offer a straightforward and easily-implemented method of collecting user information. The apache web server can easily be configured to allow them to be used and simple measures can be taken to ensure their use does not lead to any vulnerability in terms of security. The following tutorial describes how this can be done and provides a general overview of SSI.

<http://httpd.apache.org/docs/1.3/howto/ssi.html>

NB. Server-Side Includes are not supported by windows-based Microsoft server technologies for which ASP.NET is likely to provide the best alternative.

The basic structure of an SSI directive is as follows:

```
<!--#command argument="value" -->
```

For example, if the following SSI directive is placed in a web page on an SSI compatible server, it will automatically be replaced by the time and date that the page was last modified.

```
<!--#echo var="LAST_MODIFIED" -->
```

The **var="LAST_MODIFIED"** section sets a standard 'environment variable' which, in this case, is the time and date of the last modification. The **#echo** command then writes this variable to the web page, effectively replacing the directive with the value of the variable. The

whole directive is placed within HTML comments `<!-- -->` so that it will be ignored by the browser if the server fails to process it correctly.

By placing SSI directives in hidden text boxes, information can be collected and submitted alongside the form without interfering with the working of the questionnaire. The following information can be easily gathered using SSI directives:

Date and time

The date and time that the questionnaire was accessed can be established using the following SSI directive:

```
<!--#echo var="DATE_LOCAL"-->
```

If this is placed in a web page on an SSI compatible server, it will be replaced by the time and date in the format specified by the default settings of the server. This may be as follows:

Day, Date, Time, Time zone, e.g.

Saturday, 31-Dec-2005 23:59:50 GMT

To modify the format of the date, the following SSI directive can be placed immediately before `<!--#echo var="DATE_LOCAL" -->`.

```
<!--#config timefmt="value" -->
```

where **value** is replaced by a series of SSI time codes such as the following:

Code	Description	Example
%a	Abbreviated day of the week	Sat
%A	Full day of week	Saturday
%b	Abbreviated month name	Dec
%B	Full month name	December
%D	Numerical date (likely to be expressed in the Month/Day/Year format)	12/31/05
%d	Day as digit	31
%m	Month as digit	12
%H	Hour (24-Hour format)	23
%I	Hour (12-Hour format)	11
%M	Minutes	59
%S	Seconds	50
%p	AM or PM	PM
%T	24-Hour time	23:59:50
%y	2-digit year	05
%Y	4-digit year	2005
%B	Full month name	December

Thus the following directives

```
<!--#config timefmt="Date: %d/%m/%Y Time: %T" --><!--#echo var="DATE_LOCAL"-->
```

display the date and time as follows:

Date: 31/12/2005 Time: 23:59:50

This information can be added to hidden form elements for submission with the form data as follows:

```
<input type="hidden" name="date" value="<!--#config timefmt="%d/%m/%Y" --><!--#echo var="DATE_LOCAL"-->" />
```

```
<input type="hidden" name="time" value="<!--#config timefmt="%T"
--><!--#echo var="DATE_LOCAL"-->" />
```

Browser information

It is possible to use SSI directives to gather information about the browser which is being used to access the questionnaire. As stated in the introduction, this information may prove valuable in allowing an overview to the gained of the technologies available to respondents. If it becomes clear, for example, that older or less common browsers are being used to access the questionnaire, it can be tested on these browsers and redesigned if necessary.

Information about browser type and version can be established using the following directive:

```
<!--#echo var="HTTP_USER_AGENT" -->
```

When placed in a web page on an SSI compatible server, it will be replaced by a string of information about the browser's name, version, platform, and language settings. Some of the information may be confusing especially software/version information which usually begins the string and which may suggest that an older version is being used. Information from some browsers may also be more extensive than from others. However, the type and version can usually be established successfully. Some examples of the results of the directive when the most common browsers are used are as follows, with the key browser information highlighted in bold:

Internet Explorer 6

Mozilla/4.0 (compatible; **MSIE 6.0**; Windows NT 5.1; .NET CLR 1.1.4322)

Firefox 1.0.4

Mozilla/5.0 (Windows; U; Windows NT 5.1; en-GB; rv:1.7.8) Gecko/20050511 **Firefox/1.0.4**

Netscape 7.2:

Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.2) Gecko/20040804 **Netscape/7.2** (ax)

Opera 7.54

Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1) **Opera 7.54** [en]

If you wish to decipher any more unusual strings of information that may be returned, it may be necessary to refer to explanatory lists of user agents. The following is a very extensive list of browser types and typical user agent strings that they return:

http://www.zytrax.com/tech/web/browser_ids.htm

A second SSI directive which offers information about the participant's browser is as follows:

```
<!--#echo var="HTTP_ACCEPT" -->
```

This provides a list of the types of information the client will accept from the server. These are expressed as a list of MIME types in a *type/acceptable subtype* format as in the following example:

image/gif, image/jpeg, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, application/x-shockwave-flash, */*

The */* syntax indicates that the client machine will accept all types of information.

A final directive which can be used to collect browser information is as follows:

```
<!--#echo var="HTTP_ACCEPT_LANGUAGE" -->
```

This provides the language(s) that are defined as being the user's preferred language(s), as in the following example which specifies that Portuguese (Brazilian) and English (British) are the preferred languages stated by the client machine:

pt-br, en-gb

User identification

When placed in a web page on an SSI compatible server, the following directive will be replaced by the IP address of the client computer accessing the page.

```
<!--#echo var="REMOTE_ADDR" -->
```

The IP address is a group of four numbers separated by full stops (e.g. 81.77.39.115) which are associated with a particular computer or server. Collecting the IP address of the client computer that the participant used to access the questionnaire is worthwhile as repeat submissions from the same IP address within a short space of time may indicate multiple submission from the same participant and can be investigated further.

However, the use of IP addresses as the sole means of identifying multiple submissions from the same machine is unreliable as the IP address of a computer which accesses the internet via an internet-service provider is likely to be different each time it is connected to the internet. Possible multiple submissions are thus only likely to be traceable if they are carried out in a single internet session. Even when there is only a short space of time between multiple submissions from the same IP address, it may be difficult to establish whether they have come from the same user or from different users of shared computers or of computers on a shared server with the same IP address.

The use of cookies to identify particular computers offers potential for more reliable user identification, but this remains limited (see section below).

For truly effective access control, a server-side system of access via password is likely to be needed so that only invited participants with a password can access the questionnaire (see the 'Server-side processing 2' section of this technical guide.

The referring page

The following directive can be used to display the URL of the page from which a link to the questionnaire was followed:

```
<!--#echo var="HTTP_REFERER" -->
```

If the participant reached the questionnaire directly without following a link, there will be no referrer and the directive will be replaced by the following:

(none)

Data gathering via JavaScript

It is also possible to use JavaScript to collect information about the user's computer, date and time information, and the referring link (the link that the participant followed to reach the questionnaire). The following information can be gathered:

Name of browser:	<input type="text"/>
Version of browser:	<input type="text"/>
Browser platform:	<input type="text"/>
Screen size:	<input type="text"/>
Cookies enabled? (true or false)	<input type="text"/>
Referring link:	<input type="text"/>
Date:	<input type="text"/>
Time started (measured from when the page is loaded)	<input type="text"/>
Time ended (added when the form is submitted)	<input type="text"/>
Completion time	<input type="text"/>

In a questionnaire, this information could be placed within hidden form elements and submitted alongside the questionnaire data.

NB. It should be remembered that time information is measured according to the clock settings on the client machine which may not be accurate and which may mean that data is subject to time-zone differences.

However, the completion time is likely to be accurate as the start and end times will be taken from the same clock settings.

The form

To collect this information, hidden form fields are given suitable names and placed within the questionnaire form which is also given a suitable name. The information is added to the hidden form fields through the action of two functions called **startForm()** and **endTime()**.

The **startForm()** function adds the relevant information to the first eight hidden form fields when the form loads. The function is called by the **onload** event handler when the body of the HTML document loads into the browser.

The **endForm()** function adds the time of the questionnaire submission and the total completion time when the form is submitted. It is called by the **onSubmit** event handler in the form tag.

The HTML is shown below.

```
<body onload="startForm();">

<form name="quesForm" action="" method="post" onSubmit="return
endTime();">

ADD THE FOLLOWING BOXES AT THE END OF THE QUESTIONNAIRE
QUESTIONS...

<input name="browserNameBox" type="hidden" />
<input name="browserVersionBox" type="hidden" />
<input name="browserPlatformBox" type="hidden" />
<input name="screenSizeBox" type="hidden" />
<input name="enabledCookiesBox" type="hidden" />
<input name="referrerBox" type="hidden" />
<input name="dateBox" type="hidden" />
<input name="startTimeBox" type="hidden" />
<input name="endTimeBox" type="hidden" />
<input name="completionTimeBox" type="hidden" />

<input type="submit" name="Submit" value="Submit" />

</form>

</body>
```

Data collection

As you will see, the data collection functions take advantage of the following JavaScript objects and properties to collect the information:

Object&property	Description	Example
navigator.appName	Returns the name of the browser being used to access the page.	Microsoft Internet Explorer
navigator.appVersion	Returns the version of the browser.	4.0 (compatible; MSIE 6.0; Windows NT 5.1)
navigator.platform	Returns the browser's platform or operating system.	Win32
navigator.cookieEnabled	Returns a value of true if cookies are enabled and false if they are disabled.	true
screen.width	returns the width in pixels of the user's screen.	1024
screen.height	returns the height in pixels of the user's screen.	768
window.document.referrer	returns the URL of the page from which the user navigated to the current page. Returns an empty string if there is no referring link.	http://www.le.ac.uk

date.getDate()	Returns the day of the month	25
date.getMonth()	Return the month of the year as a number from 0 (January) to 11 (December)	11
date.getFullYear()	Returns the year (may not work in older browsers)	2005
date.getHours()	Returns the hour of the day (24-Hour format)	23
date.getMinutes()	Returns the minute of the hour	59
date.getSeconds()	Returns the second of the minute	50
date.getTime()	Returns the current time in milliseconds	112473628262

The information is automatically returned when the objects and properties are included in the code.

The JavaScript functions

The code for the first function is as follows:

```
function startForm(){
// Change the value of the hidden form field called
'browserNameBox' to the value returned by the 'appName' property
of the 'navigator' object (i.e. the name of the browser used to
access the form)
document.infoForm.browserNameBox.value = navigator.appName;
// Change the value of the browserVersionBox
document.infoForm.browserVersionBox.value = navigator.appVersion;
// Change the value of the browserPlatformBox
document.infoForm.browserPlatformBox.value = navigator.Platform;
// Change the value of the enabledCookiesBox
document.infoForm.enabledCookiesBox.value =
navigator.cookieEnabled;
// Change the value of the screenSizeBox by adding the values of
screen.width and screen.height separated by a string (" x ")
document.infoForm.screenBox.value = screen.width + " x " +
screen.height;
// Change the value of the referrerBox
document.infoForm.referrerBox.value = window.document.referrer;
// Change the value of the dateBox
// Declare a new date object given the name 'date'
var date = new Date();
// Set a variable called 'day' to store the day of the month of
this date object
var day = date.getDate();
```



```
// Format the day by adding a zero if it is lower than 10 so that  
the fifth will appear as 05 rather than 5
```

```
if (day < 10){  
day = "0" + day;  
}
```

```
// Set a variable called 'month' to store the month of this date  
object and format it so that January appears as 01 rather than 0  
and December appears as 12 rather than 11
```

```
var month = date.getMonth() + 1;  
if (month < 10){  
month = "0" + month;  
}
```

```
// Set a variable called 'year' to store the year of this date  
object
```

```
var year = date.getFullYear();
```

```
// Add the full date to the hidden form field called dateBox by  
adding the variables 'day', 'month' and 'year' separated by  
strings ("/")
```

```
document.infoForm.dateBox.value = day + "/" + month + "/" + year;
```

```
// Change the value of the startTimeBox by setting a new date  
object and storing the hours, minutes and seconds of this date  
object in variables. In each case, format the time so that digits  
lower than 10 are preceded by a zero
```

```
var startTime = new Date();  
var hour = startTime.getHours();  
if (hour < 10){  
hour = "0" + hour;  
}  
var minute = startTime.getMinutes();  
if (minute < 10){  
minute = "0" + minute;  
}  
var second = startTime.getSeconds();  
if (second < 10){  
second = "0" + second;  
}
```

```
// Add the complete time to the 'startTimeBox' hidden form field  
by adding the variables 'hour', 'minute' and 'second' separated  
by strings (" : ")
```

```
document.infoForm.startTimeBox.value = hour + " : " + minute +  
" : " + second;
```

```
// Finally set a variable called 'startTiming' with the current time in milliseconds. This will be used to calculate completion time.
```

```
now = new Date();  
startTiming = (now.getTime())  
}
```

The code for the second function is as follows:

```
function endForm(){
```

```
// Get the current time in the same way as in the previous function
```

```
var endTime = new Date();  
var hour = endTime.getHours();  
if (hour < 10){  
hour = "0" + hour;  
}  
var minute = endTime.getMinutes();  
if (minute < 10){  
minute = "0" + minute;  
}  
var second = endTime.getSeconds();  
if (second < 10){  
second = "0" + second;  
}
```

```
// Add the complete time to the hidden form field 'endTimeBox'
```

```
document.infoForm.endTimeBox.value = hour + " : " + minute + " : "  
+ second;
```

```
// Set a variable called 'endTiming' with the current time in milliseconds.
```

```
now = new Date();  
endTiming = (now.getTime());
```

```
// Calculate completion time by subtracting the start time obtained by the first function (called when the page loaded) from this end time (calculated at submission). Round up the result and convert it from milliseconds to seconds. Store it in a variable called 'timeTaken'.
```

```
timeTaken=Math.round((endTiming-startTiming)/1000);
```

```
// Add this to the hidden form field 'completionTimeBox' followed by the string " secs"
```

```
document.infoForm.completionTimeBox.value = timeTaken + " Secs";
```

```
// Finally, return true to the form to allow submission to proceed
```

```
return true;
}
```

The complete HTML and code is shown below. There are also instructions on how to incorporate the hidden form fields and code into your own questionnaires.

```
<html>
<head>
<title>Exploring ORMs | Example of collecting user information
via JavaScript</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-
8859-1">

<link href="../../../generic/nn4.css" rel="stylesheet" type="text/css">
<style type="text/css">
@import url(../generic/main.css);
</style>

<script language="javascript" type="text/javascript">
function startForm(){
// Change the value of the hidden input fields that contain
information gathered when the form is loaded

document.infoForm.browserNameBox.value = navigator.appName;
document.infoForm.browserVersionBox.value = navigator.appVersion;
document.infoForm.browserPlatformBox.value = navigator.platform;
document.infoForm.enabledCookiesBox.value =
navigator.cookieEnabled;
document.infoForm.screenSizeBox.value = screen.width + " x " +
screen.height;
document.infoForm.referrerBox.value = window.document.referrer;

// Get date and time info and change the value of the appropriate
hidden form fields

var date = new Date();
var day = date.getDate();
if (day < 10){
day = "0" + day;
}
var month = date.getMonth() + 1;
if (month < 10){
month = "0" + month;
}
var year = date.getFullYear();
document.infoForm.dateBox.value = day + "/" + month + "/" + year;

var startTime = new Date();
var hour = startTime.getHours();
if (hour < 10){
hour = "0" + hour;
}
var minute = startTime.getMinutes();
```

```

if (minute < 10){
minute = "0" + minute;
}
var second = startTime.getSeconds();
if (second < 10){
second = "0" + second;
}
document.infoForm.startTimeBox.value = hour + " : " + minute +
" : " + second;

// Get the current time in milliseconds.

now = new Date();
startTiming = (now.getTime())
}

function endForm(){

// Get the current time and add to the appropriate form field

var endTime = new Date();
var hour = endTime.getHours();
if (hour < 10){
hour = "0" + hour;
}
var minute = endTime.getMinutes();
if (minute < 10){
minute = "0" + minute;
}
var second = endTime.getSeconds();
if (second < 10){
second = "0" + second;
}
document.infoForm.endTimeBox.value = hour + " : " + minute + " :
" + second;

// Get the current time in milliseconds, subtract the start time
obtained by the first function from this end time and add the
result to the form field

now = new Date();
endTiming = (now.getTime());
timeTaken=Math.round((endTiming-startTiming)/1000);
document.infoForm.completionTimeBox.value = timeTaken + " Secs";

// Produces alert box to show the contents of the hidden boxes at
submission - delete from final version
alert("browserNameBox.value = " +
document.infoForm.browserNameBox.value +
"\rbrowserVersionBox.value = " +
document.infoForm.browserVersionBox.value +
"\rbrowserPlatformBox.value = " +
document.infoForm.browserPlatformBox.value +
"\renabledCookiesBox.value = " +

```

```

document.infoForm.enabledCookiesBox.value +
"\rscreenSizeBox.value = " +
document.infoForm.screenSizeBox.value +
"\rreferrerBox.value = " + document.infoForm.referrerBox.value +
"\rdateBox.value = " + document.infoForm.dateBox.value +
"\rstartTimeBox.value = " + document.infoForm.startTimeBox.value
+
"\rendTimeBox.value = " + document.infoForm.endTimeBox.value +
"\rcompletionTimeBox.value = " +
document.infoForm.completionTimeBox.value);

// Return true to the form to allow submission to proceed
return true;
}
</script>
</head>
<body onload = "startForm()">
<form name="infoForm" action="" method="post" onSubmit="return
endForm();">
<div class="ques">
<p>The form begins here</p>
<p>&nbsp;</p>
<p>Add your questionnaire questions here...</p>
<p>&nbsp;</p>
<p>The hidden boxes are placed below (See page source). In this
example, alert
boxes have been added to show the contents of the boxes when the
page is
submitted so that the effect of the script can be observed and
checked.
You should delete the sections of the script that produce these
alert boxes
from the final version of the form.</p>
<input name="browserNameBox" type="hidden" />
<input name="browserVersionBox" type="hidden" />
<input name="browserPlatformBox" type="hidden" />
<input name="screenSizeBox" type="hidden" />
<input name="enabledCookiesBox" type="hidden" />
<input name="referrerBox" type="hidden" />
<input name="dateBox" type="hidden" />
<input name="startTimeBox" type="hidden" />
<input name="endTimeBox" type="hidden" />
<input name="completionTimeBox" type="hidden" />
<p>
<input name="submit" type="submit" value="SUBMIT"/>
</p>
<p>&nbsp;</p>

```

```

<p>If you are using validation code in a function called by the
onSubmit event
handler (see the 'Form validation' section), the
<code>endForm</code> function can
be called from within this validation function. If the validation
routine
finds that the form is ready to be submitted it will <code>return
true</code>
to the form to allow submission to proceed. By adding
<code>endForm()</code>
to call the function directly before the <code>endForm()</code>
syntax,
the hidden form fields will be changed before the final
submission. </p>
<p>The final section of the validation code will thus be </p>
<div class = "codeborder">
<p><code>&nbsp;&nbsp;&nbsp;endForm()</code>;<br>
<code>&nbsp;&nbsp;&nbsp;endForm()</code>;</p>
</div>
<p>rather than simply </p>
<div class = "codeborder">
<p><code>&nbsp;&nbsp;&nbsp;return true</code>;</p>
</div>
<p></p>
<p>Select the following link to see a <a href="webform7.htm">web
form with
the information gathering functions combined with validation
routines</a>.</p>
</div>

</form>
</body>
</html>

```

Combining data gathering with validation

If you are using validation code in a function called by the **onSubmit** event handler, the **endForm** function cannot be called from within the form at the same time. Instead, it should be called from within the validation function itself.

If the validation routine finds that the form is ready to be submitted it will **return true** to the form to allow submission to proceed (see the 'Form validation' section). By adding **endForm()** to call the function before the **return true** syntax, the hidden form fields will be changed before the final submission.

The final lines of the section of the validation code that deals with successfully completed forms will thus be

```

endForm();
return true;

```

rather than simply

```
return true;
```

The following HTML and code can be used to create a web form with the information gathering functions combined with validation routines:

```
<html>
<head>
<title>Exploring ORMs | Example of validating multiple form
elements</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-
8859-1">
<link href="../generic/nn4.css" rel="stylesheet" type="text/css">
<style type="text/css">
@import url(../generic/main.css);
</style>
<script language="javascript" type="text/javascript">

// Set variables

// Variable for whether submit has been pressed

var submitPressed = false;

// Variable for each question

var q1Answered = false;
var q2Answered = false;

// Variable for the final alert box message

var alertMessage = "";

// start function

function checkForm(form) {

// Check question 1 - If the box is empty, the question has not
been answered. In this case, add the question number to the
message that will be delivered to the user to indicate that there
are problems with certain questions.

if (form.namebox.value == ""){
alertMessage = "1 ";
}

// If the box is not empty, the question has been answered, so
the 'q1Answered' variable is changed to 'true'.

else{
q1Answered = true;
}

// Check question 2 - If any of the buttons have been selected,
change 'q2Answered' variable to 'true'.
```

```

for (var i = 0; i<form.often.length; i++){
if (form.often[i].checked == true){
q2Answered = true;
}
}

// If 'q2Answered' is still false, the question has not been
answered, so add the question number to the message.
if (q2Answered == false){
alertMessage = alertMessage + "2 ";
}

// If both questions have been answered (q1Answered and
q2Answered are both true) or if the user has already tried to
submit once before (submitPressed is true), provide the user with
a thank-you alert and return 'true' to the form to allow
submission to proceed.

if (submitPressed == true || (q1Answered == true && q2Answered ==
true)){
alert("Thank you for completing the form.");
form.namebox.value = ""
submitPressed = false;
q1Answered = false;
q2Answered = false;
alertMessage = "";
// Call the endForm() function to collect timing information
before the form is submitted.
endForm();
return true;
}

// If either or both of the questions has not been answered,
deliver an alert box message to the user and include the variable
'alertMessage' which was used to collect the numbers of the
unanswered questions. Change the textBoxSubmitPressed variable to
true to allow submission to proceed next time and return 'false'
to the form to prevent submission this time.

else {
alert("Please check your answers to the following
question(s):\n\r" + alertMessage + "\n\rIf you are happy with
them, press the submit button again to proceed with the
submission.");
submitPressed = true;
return false;
}
}

function startForm(){

// Change the value of the hidden input fields that contain
information gathered when the form is loaded

```



```

document.infoForm.browserNameBox.value = navigator.appName;
document.infoForm.browserVersionBox.value = navigator.appVersion;
document.infoForm.browserPlatformBox.value = navigator.platform;
document.infoForm.enabledCookiesBox.value =
navigator.cookieEnabled;
document.infoForm.screenSizeBox.value = screen.width + " x " +
screen.height;
document.infoForm.referrerBox.value = window.document.referrer;

// Get date and time info and change the value of the appropriate
hidden form fields

var date = new Date();
var day = date.getDate();
if (day < 10){
day = "0" + day;
}
var month = date.getMonth() + 1;
if (month < 10){
month = "0" + month;
}
var year = date.getFullYear();
document.infoForm.dateBox.value = day + "/" + month + "/" + year;

var startTime = new Date();
var hour = startTime.getHours();
if (hour < 10){
hour = "0" + hour;
}
var minute = startTime.getMinutes();
if (minute < 10){
minute = "0" + minute;
}
var second = startTime.getSeconds();
if (second < 10){
second = "0" + second;
}
document.infoForm.startTimeBox.value = hour + " : " + minute +
" : " + second;

// Get the current time in milliseconds.

now = new Date();
startTiming = (now.getTime())
}

function endForm(){

// Get the current time and add to the appropriate form field

var endTime = new Date();
var hour = endTime.getHours();
if (hour < 10){
hour = "0" + hour;

```

```

}
var minute = endTime.getMinutes();
if (minute < 10){
minute = "0" + minute;
}
var second = endTime.getSeconds();
if (second < 10){
second = "0" + second;
}
document.infoForm.endTimeBox.value = hour + " : " + minute + " : " + second;

// Get the current time in milliseconds, subtract the start time
obtained by the first function from this end time and add the
result to the form field

now = new Date();
endTiming = (now.getTime());
timeTaken=Math.round((endTiming-startTiming)/1000);
document.infoForm.completionTimeBox.value = timeTaken + " Secs";

// Produces alert box to show the contents of the hidden boxes at
submission - delete from final version
alert("browserNameBox.value = " +
document.infoForm.browserNameBox.value +
"\rbrowserVersionBox.value = " +
document.infoForm.browserVersionBox.value +
"\rbrowserPlatformBox.value = " +
document.infoForm.browserPlatformBox.value +
"\renabledCookiesBox.value = " +
document.infoForm.enabledCookiesBox.value +
"\rscreenSizeBox.value = " +
document.infoForm.screenSizeBox.value +
"\rreferrerBox.value = " + document.infoForm.referrerBox.value +
"\rdateBox.value = " + document.infoForm.dateBox.value +
"\rstartTimeBox.value = " + document.infoForm.startTimeBox.value
+
"\rendTimeBox.value = " + document.infoForm.endTimeBox.value +
"\rcompletionTimeBox.value = " +
document.infoForm.completionTimeBox.value);

// Return true to the form to allow submission to proceed
return true;
}
</script>

</head>

<body onload = "startForm()">

<form name="infoForm" action="" method="post" onSubmit="return
checkForm(this);">

```

```

<div class="ques">
<p>1. What is your name?</p>
<p><input type="text" name="namebox" /></p>
<p>2. How often do you use the internet?</p>
<p>
<input type="radio" name="often" value="everyday" />
everyday<br />
<input type="radio" name="often" value="2-3 days per week" />
2-3 days per week<br />
<input type="radio" name="often" value="4-5 days per week" />
4-5 days per week<br />
<input type="radio" name="often" value="6-7 days per week" />
6-7 days per week<br />
<input type="radio" name="often" value="less than once a week" />
less than once a week</p>
<p>The hidden boxes are placed below (See page source). In this
example, alert
boxes have been added to show the contents of each box when the
page is
loaded and submitted so that the effect of the script can be
observed and
checked. You should delete the sections of the script that
produce these
alert boxes from the final version of the form.</p>
<input name="browserNameBox" type="hidden" />
<input name="browserVersionBox" type="hidden" />
<input name="browserPlatformBox" type="hidden" />
<input name="screenSizeBox" type="hidden" />
<input name="enabledCookiesBox" type="hidden" />
<input name="referrerBox" type="hidden" />
<input name="dateBox" type="hidden" />
<input name="startTimeBox" type="hidden" />
<input name="endTimeBox" type="hidden" />
<input name="completionTimeBox" type="hidden" />
<p><input type="submit" name="Submit" value="Submit" /></p>
</div>
</form>
</body>
</html>

```

Dealing with non-JavaScript browsers

Providing that testing is carried out to ensure that the JavaScript used will not interfere with the completion and submission of the questionnaire in non-JavaScript browsers, there will be no problems for respondents without JavaScript. The hidden form fields will simply be returned with empty values.

A simple way of gathering the information that JavaScript was not available (and verifying that empty values are caused by this rather than by any problems with the form or the scripting) is to include a hidden form field enclosed in `<noscript></noscript>` tags as follows.

```
<noscript><input type="hidden" name="JavaScript"
value="unavailable"></noscript>
```

This will only add the code within the tags in cases where JavaScript is not available so that the value will only be sent along with the form if this is the case.

Cookies

Setting a cookie to identify when a computer has already been used to submit a questionnaire is more reliable than using IP addresses. When a user submits the questionnaire, a cookie can be stored on the hard drive of the computer which will identify the fact that it has been used to make a submission. Should the same computer be used to access the questionnaire a second time, action can be taken such as delivering a message or redirecting the user.

However, problems remain in the use of this method to prevent multiple submission as users can opt to reject cookies or delete them from their hard drive. These examples also depend on the use of JavaScript which can be disabled in the browser. It should also be remembered that different participants may be using the same machine to access the questionnaire.

For truly effective access control, a system of access via password is likely to be needed so that only invited participants with access to a password can participate (see the 'Server-side processing' section of this technical guide).

Where the researcher does not wish to limit access in this way, however, the use of cookies is likely to be helpful in highlighting or preventing multiple submissions as the majority of users are unlikely to have disabled cookies on their machines. The process of identifying when a computer has already been used to submit a questionnaire using cookies is straightforward.

Examples

In the following examples, a pop-up window with a questionnaire is opened when the user clicks on a link. When the page is opened, JavaScript is used to check whether or not the computer has a cookie called "Multiple" saved on it. If it does not, it is assumed that it is the first time that the computer has been used to access the questionnaire. A cookie called "Multiple" is then saved to ensure that the first check will establish that the computer has been used to access the questionnaire if it is subsequently used again.

Each of the examples carries out a different action if the cookie called "Multiple" is found. The first displays an alert-box message. The second reveals the same message at the top of the page. The third redirects the user to a new page which explains that the computer has been used to access the questionnaire before and provides a link to a different version of the questionnaire.

Explanation

The JavaScript code for the first example is as follows:

```
<script language="javascript" type="text/javascript">
function checkCookie(){
var found;
if (document.cookie.length>0){
```

```

found = document.cookie.indexOf("multiple=");
if (found != -1){
alert("It appears that your computer has been used to access this
questionnaire before. If it was used by you, please do not
complete the questionnaire a second time.")
return;
}
}
setCookie("multiple", "yes", "");
}
function setCookie(name, value, expires){
if (expires == ""){
var now = new Date();
now.setTime(now.getTime() + (1000*60*60*24*365));
expires = now.toGMTString();
}
document.cookie = escape(name) + "=" + escape(value) + "; path
=/" + ";expires=" + expires;
}
}
</script>

```

The first function `checkCookie()` checks whether or not the cookie called **"multiple"** has been previously saved to the computer's hard drive. If it has, the alert message is delivered. If not, the `setCookie("multiple", "yes", "")` function is called. This saves a cookie called **"multiple"** with the value **"yes"**, and sets the expiry date for the cookie to 1 year from the current date ($1000*60*60*24*365$) which is the number of hours ($1000*60*60$) multiplied by the hours in a day and the days in a year ($24*365$).

The `checkCookie()` function is then called when the body of the page loads using the `onload` event handler:

```
<body onload="checkCookie();">
```

The code for the other examples is similar, but the action that occurs when the cookie called **"multiple"** is found is different.

For the second example, a function called `showMessage('multiple')` is called which shows the hidden layer called **'multiple'** to reveal the message at the top of the page. The `setCookie()` function is the same, but the `checkCookie()` function changes as follows:

```

function checkCookie(){
var found;
if (document.cookie.length>0){

found = document.cookie.indexOf("multiple=");
if (found != -1){
showMessage('multiple');
return;
}
}

```

```
}  
setCookie("multiple", "yes", "");  
}
```

The following functions are also added to provide the functionality required to reveal the message at the top of the questionnaire.

```
// The message to the participant is delivered by showing or  
hiding layers. This is done differently according to the type of  
browser the participant is using. This block of code checks the  
type of browser
```

```
var isIE4 = false;  
var isCompliant = false;  
if(document.getElementById){  
  
if(!document.all)  
{  
isCompliant=true;  
}  
if(document.all)  
{  
isIE4=true;  
}  
}  
}
```

```
// Code to allow the hidden layer used to deliver the messages to  
be made visible. A different method is used according to the type  
of browser identified in the previous block of code.
```

```
function aLs(layerID){  
var returnLayer = "null";  
if(isIE4){  
  
returnLayer = eval("document.all." + layerID + ".style");  
}  
  
if(isCompliant){  
  
returnLayer = eval("document.getElementById('" + layerID +  
"').style");  
}  
}  
return returnLayer;  
}
```

```
// Function to make a layer visible when called. The layer name  
is fed into the function when it is called.
```

```
function showMessage(ID)  
{  
aLs(ID).display = "";  
}
```

For the third example, a the URL of the page is changed using the code `window.document.location="newpage.htm"`; where `"newpage.htm"` is the link to the page called when the cookie called "Multiple" is found. The `setCookie()` function is the same, but the `checkCookie()` function changes as follows:

```
function checkCookie(){
var found;
if (document.cookie.length>0){

found = document.cookie.indexOf("multiple=");
if (found != -1){
window.document.location="newpage.htm";
return;
}
}
setCookie("multiple", "yes", "");
}
```

This URL `"newpage.htm"` is automatically loaded if the cookie is found. This page includes a link to a different version of the questionnaire which can be submitted using a hidden form field to alert the researcher that the computer had already been used to complete the questionnaire and prompt him/her to check further (see the 'Adding data to hidden form fields' section above).

Complete code

The complete HTML and JavaScript for each of the three examples shown in this section can be seen below. The code can be saved, or copied and pasted into your text editor.

Example 1: An alert box message

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Exploring ORMs | Technical guide | Cookies: Example
1</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-
8859-1" />
<link href="../generic/nn4.css" rel="stylesheet" type="text/css">
<style type="text/css">
@import url(../generic/main.css);
</style>

<script language="javascript" type="text/javascript">
function checkCookie(){
var found;
if (document.cookie.length>0){
found = document.cookie.indexOf("multiple=");
if (found != -1){
alert("It appears that your computer has been used to access this
questionnaire before. If it was used by you, please do not
```

```

complete the questionnaire a second time.")
return;
}
}
setCookie("multiple", "yes", "");
}
function setCookie(name, value, expires){
if (expires == ""){
var now = new Date();
now.setTime(now.getTime() + (1000*60*60*25*365));
expires = now.toGMTString();
}
document.cookie = escape(name) + "=" + escape(value) + "; path
=/" + ";expires=" + expires;
}
function deleteCookie(){
document.cookie = "multiple" + ";; expires = Thu, 01-Jan-70
00:00:01 GMT" + "; path=/"
}
}
</script>
</head>

<body onload="checkCookie();">
<div class="ques">

<h1>Welcome to the questionnaire</h1>
<p>1. What is your name?</p>

<p><input type="text" name="namebox" /></p>

<p>2. How often do you use the internet?</p>

<p><input type="radio" name="often" value="everyday"
/>everyday<br />
<input type="radio" name="often" value="2-3 days per week" />2-3
days per week<br />
<input type="radio" name="often" value="4-5 days per week" />4-5
days per week<br />
<input type="radio" name="often" value="6-7 days per week" />6-7
days per week<br />
<input type="radio" name="often" value="less than once a week"
/>less than once a week</p>

<p><input type="submit" name="Submit" value="Submit" /></p>
</div>
</body>
</html>

```

Example 2: A message appears at the top of the page

```

<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

```



```

<head>
<title>Exploring ORMs | Technical guide | Cookies: Example
2</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-
8859-1" />
<link href="../generic/nn4.css" rel="stylesheet" type="text/css">
<style type="text/css">
@import url(../generic/main.css);
</style>
<style type ="text/css">
.message {
font-weight: bold;
color: #f00;
}
</style>
<script language="javascript" type="text/javascript">
function checkCookie(){
var found;
if (document.cookie.length>0){
found = document.cookie.indexOf("multiple=");
if (found != -1){
showMessage('multiple');
return;
}
}
setCookie("multiple", "yes", "");
}
function setCookie(name, value, expires){
if (expires == ""){
var now = new Date();
now.setTime(now.getTime() + (1000*60*60*25*365));
expires = now.toGMTString();
}
document.cookie = escape(name) + "=" + escape(value) + "; path
=/" + ";expires=" + expires;
}

var isIE4 = false;
var isCompliant = false;
if(document.getElementById){

if(!document.all)
{
isCompliant=true;
}
if(document.all)
{
isIE4=true;
}
}
}

```

```

function aLs(layerID){
var returnLayer ="null";
if(isIE4){
returnLayer = eval("document.all." + layerID + ".style");
}
if(isCompliant){
returnLayer = eval("document.getElementById('" + layerID +
"').style");
}
}
return returnLayer;
}

function showMessage(ID)
{
aLs(ID).display = "";
}
</script>
</head>

<body onload="checkCookie();">

<form name="form1"><div class="ques">
<div id="multiple" style="display:none">
<div class="message">It appears that your computer has been used
to access
this questionnaire before. If it was used by you, please do not
complete
the questionnaire a second time.</div>
</div>
<h1>Welcome to the questionnaire</h1>
<p>1. What is your name?</p>

<p><input type="text" name="namebox" /></p>

<p>2. How often do you use the internet?</p>

<p><input type="radio" name="often" value="everyday"
/>everyday<br />
<input type="radio" name="often" value="2-3 days per week" />2-3
days per week<br />
<input type="radio" name="often" value="4-5 days per week" />4-5
days per week<br />
<input type="radio" name="often" value="6-7 days per week" />6-7
days per week<br />
<input type="radio" name="often" value="less than once a week"
/>less than once a week</p>

<p><input type="submit" name="Submit" value="Submit" /></p>
</div></form>

```

```
</body>
</html>
```

Example 3: The user is redirected to another page

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Exploring ORMs | Technical guide | Cookies: Example
3</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-
8859-1" />
<link href="../generic/nn4.css" rel="stylesheet" type="text/css">
<style type="text/css">
@import url(../generic/main.css);
</style>
<script language="javascript" type="text/javascript">
window.opener.close();
function checkCookie(){
var found;
if (document.cookie.length>0){
found = document.cookie.indexOf("multiple=");
if (found != -1){
window.document.location="cookies3b.htm";
return;
}
}
setCookie("multiple", "yes", "");
}
function setCookie(name, value, expires){
if (expires == ""){
var now = new Date();
now.setTime(now.getTime() + (1000*60*60*25*365));
expires = now.toGMTString();
}
document.cookie = escape(name) + "=" + escape(value) + "; path
=/" + ";expires=" + expires;
}
</script>
</head>

<body onload="checkCookie();">

<form name="form1"><div class="ques">
<h1>Welcome to the questionnaire</h1>
<p>1. What is your name?</p>

<p><input type="text" name="namebox" /></p>

<p>2. How often do you use the internet?</p>
```

```
<p><input type="radio" name="often" value="everyday"
/>everyday<br />
<input type="radio" name="often" value="2-3 days per week" />2-3
days per week<br />
<input type="radio" name="often" value="4-5 days per week" />4-5
days per week<br />
<input type="radio" name="often" value="6-7 days per week" />6-7
days per week<br />
<input type="radio" name="often" value="less than once a week"
/>less than once a week</p>

<p><input type="submit" name="Submit" value="Submit" /></p>
</div></form>
</body>
</html>
```

Server-side processing

Introduction

Once a questionnaire has been designed and created, it is necessary to make it available online and add a means of collecting and processing the results. Doing this successfully requires the addition of server-side processes.

This page is designed to provide you with an overview of server-side processing technologies. It will explain what server-side processing is and how it can be used to collect data from questionnaires. It will cover the following areas:

- The nature of server-side processing and how it differs from client-side processing;
- How server-side and client side processing interact when a questionnaire is requested and submitted;
- An overview of the options for hosting the questionnaire on a server, which will be needed to make the questionnaire available online;
- An introduction to some of the key technologies required and the options available:
 - Server software: Apache and Microsoft;
 - Server-side processing software: CGI, ASP.NET, PHP, ColdFusion and JSP;
 - Database software: MySQL and SQL Server;
- How to set up a test-server on a personal computer to use when developing using these technologies;
- An overview of how WYSIWYG web editors and other software can be used to help with the development;
- An outline of common server-side processing tasks that are carried out to process a questionnaire;
- Links to key resources on each of the server-side processing technologies.

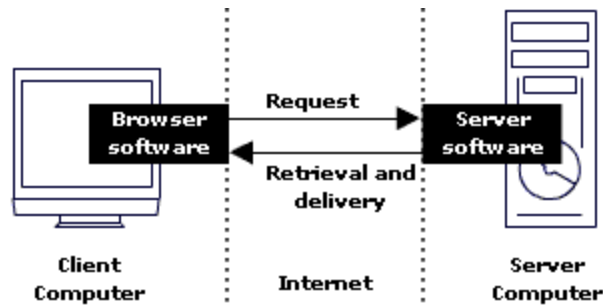
Server-side v client-side processing

To understand how server-side processing works and how it differs to client-side processing, it is useful to firstly consider what happens when a user successfully calls a web page. The main steps can be seen as follows:

1. The user types the URL of the webpage into the browser's location bar or clicks on a link. This URL acts as a reference to the computer on which the webpage is held (the server) and the location of the file (its name and position inside the file tree).
2. The browser sends a message over the internet to the server computer requesting the webpage.
3. Server software (e.g. Apache) on the server computer receives the message.
4. The URL is correct and up-to-date, and there are no problems with the server computer, so the software is able to locate the file in the computer's hard drive.
5. The software sends a copy of the file back to the browser on the requesting computer (the client).
6. The browser displays the page according to the HTML and CSS instructions it contains.

- Alternatively, if there is a problem with the server or the URL is incorrect, an error message is displayed by the browser on the client machine.

This can be shown diagrammatically as follows:



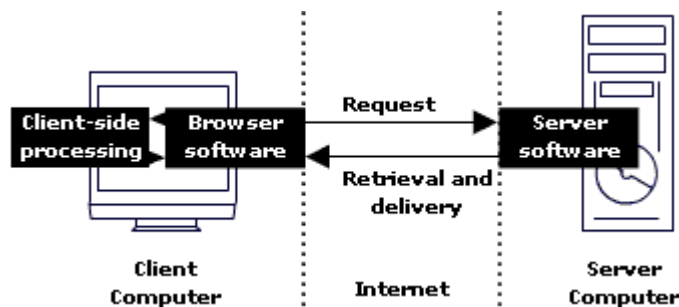
The interaction between the client and the server when a user requests a web page

The files held on the server are 'static' which means that all users who request the webpage will be delivered exactly the same content from the hard drive. It cannot be altered unless the developer makes a change to the files and saves them again.

The way that client-side and server-side processing works inside the interaction between client and server can be seen below. This is followed by an overview of the advantages and disadvantages of each.

Client-side processing

Where client-side processing is used, effects can be added which may change the look or content of the webpage. For example, JavaScript can be added to check for the presence of a cookie indicating that a user has visited the site before. If a cookie is found, an alternative message can be displayed. Alternatively, users can display or hide content such as help windows or menu bars by clicking on buttons or links. However, the content delivered by the server is always the same. Like the CSS and/or HTML it is included with, the JavaScript is 'static' and is delivered within a static page. The processing that takes place occurs through instructions to the browser on the client machine. This can be seen in the following diagram:

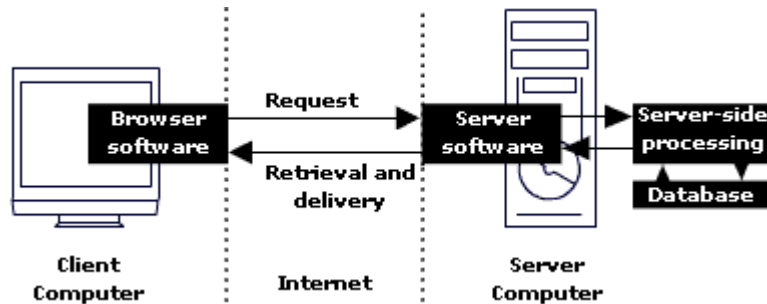


The interaction between the client and the server when a user requests a web page containing client-side processing

Server-side processing

Unlike the case of static websites, a typical use for server-side processing is to deliver 'dynamic' content to users according to choices they make before requesting the page. Thus, two visits to the same page may result in completely different content being delivered. For example, users of a bookstore site may search for particular types of books and be delivered only the results that are relevant to the search. Alternatively, server-side processing can be

used to detect information about users' computers such as screen size, type of browser or whether JavaScript is enabled, and then to send out a different page accordingly. The client computer receives the webpage from the server in exactly the same way as described above, but processing takes place to create the content of a particular page before it is sent back to the client. This may include interaction with a database to 'collect' content for display. This can be seen in the following diagram:



The interaction between the client and the server when a user requests a web page containing server-side processing

Advantages and disadvantages

Advantages of server-side processing

The main advantage of server-side processing is that it is not dependent on the client computer. It will take place regardless of the type of browser used as it occurs on the server before the page is delivered. This means that the use of server-side processing is not likely to reduce the accessibility of a webpage as the functionalities it provides should work on all platforms and with any browser. On the other hand, client-side processing depends on the availability of technologies on the client computer such as a browser which is able to process JavaScript and which is up-to-date enough to deal with the JavaScript version used. Most browsers also provide users with an option to deactivate JavaScript. Where the functionality provided by the processing is essential, server-side processing is thus the only viable option.

Server-side processing also has related security advantages in that it occurs before the page is delivered and is thus not 'visible' to the end-user, who is only given the results of the processing. Providing security measures are followed to prevent hacking of the server, these processes cannot be accessed. However, client-side processes must be delivered to the user along with the CSS and/or HTML, images and other files that make up the web page. They can thus be accessed by the user (and potentially misused or manipulated) simply by viewing the source code. Where password control is required, for example, server-side processing is the only viable option as the list of acceptable passwords and the process of checking that a valid password has been entered occurs on the server. If the process was carried out client-side, viewing the source code would immediately reveal this to the user.

Disadvantages of server-side processing

The main disadvantage of server-side processing is that it has the potential to reduce the speed and efficiency of a webpage. Every time a process needs to be carried out based on user input or actions, a call must be made to the server and the results delivered back to the client. In cases where the functionalities provided by processing are not essential and where there are no security implications, client-side processing can provide a faster and more seamless experience for the user. Thus, for example, the drop-down headings used by this site are activated client-side which adds an 'extra' functionality to the site but which is not essential to accessing the information the headings contain. Using server-side processing for such features would be impractical and slow, and if the user does not have a browser with client-side functionalities this will not affect accessibility.

Server-side processing also obviously requires access to a server which can deal with the processing technology chosen, along with adequate permissions to add scripts and/or databases to this server. Where this is not available, it may be necessary to use client-side processing, or to rethink whether the functionality should be added.

Server and Client-side processing within a questionnaire

In delivering, receiving and processing a questionnaire, the main steps in the interaction between server and client are typically as follows:

- The browser on the client computer requests the webpage containing the questionnaire from the server, receives the questionnaire and displays it on the page (see details in stages 1-6 above).
- Extra information such as the date and time, and details about the participant's computer are collected and placed in hidden form fields using client-side and/or server-side processes (see 'collecting participant information').
- The participant completes and submits the questionnaire and if JavaScript is available in the participant's computer, client-side validation occurs to check the responses (see 'Form validation').
- If JavaScript is available and there are problems with the form, submission is prevented and a message is displayed to highlight what the problems were (return to step 3).
- If JavaScript is deactivated in the participant's computer or the answers have been entered correctly, submission goes ahead and the contents of the form fields in the questionnaire are sent to the server (along with any extra information in the hidden form fields).
- Server-side validation routines are carried out to check for problems with the form.
- If the answers have been entered correctly, a thank-you message is displayed and server-side processes are carried out to add the answers to a database or text-file, and/or include them in an email to be sent to the researcher.
- If the server-side validation reveals a problem with the form, it is returned to the participant with the answers preserved and messages to highlight what the problems were. The participant corrects the problems (if so inclined) and resubmits (return to step 3).

Both server and client-side processing are used to process the questionnaire. Client-side processing is used for validation as it will provide improved usability and speed to those who have JavaScript available on their computers, but will not reduce accessibility for those who do not. Server-side processing is used to deal with the results and also to provide 'back-up' validation where JavaScript is unavailable or where users may have deactivated it to avoid validation.

Hosting options

A number of technologies are available to add server-side processing functionalities to web pages, but whichever technology is chosen, the questionnaire will need to be uploaded to a server along with the required databases and/or scripts. The choice of which technology to use is likely to depend primarily on the type of server and software available and the level of access the researcher has to this server.

The following three hosting options are likely to be available to researchers who wish to create and upload their own questionnaires:

Access to server space on an institution's server

For researchers working within an institution such as a university, a government department, an NHS trust, or a charitable organisation, it may be possible to upload the questionnaire to the institution's server. In this case it will be necessary to find out what type of processing and database software can be used, and whether there are any limitations or conditions on how they can be used. It may be that certain types of scripts are not allowed or that time is needed for computer services to check any scripts before they are uploaded. In some cases, access to the server may only be possible if standard procedures for implementing web forms are followed. For a questionnaire with 'standard' requirements, this should not be a problem but it may be necessary to check issues such as whether the results can be accessed in the required format or whether they can be emailed, automatically added to a database, or both.

Use of a commercial internet hosting service

If a researcher does not have access to a server or if the level of control over what can be done and how is not adequate, it may be necessary to use a commercial hosting service. There is a huge range of options available at a range of prices. There are many free services available, but access to server-side processing capabilities and a database is likely to only be available through services which charge. If a free service is found that seems to offer the required functionalities it is also important to check whether advertisements are included and, if so, whether this is acceptable.

Services that charge can range from relatively inexpensive options which run a number of websites on a single server, to more expensive options with servers dedicated to a particular customer and frequently targeted at the business world. When choosing a provider, it is important to be clear on what the requirements are in terms of file space and available features to make comparison easier and to find a good value provider. It is also important to be aware that there may be limits of the capabilities that are offered and that, for example, some services may only allow standard scripts to be used, rather than allowing users to upload their own scripts. Depending on what is required, this may not be a problem, but it is important to check when deciding which service to use. Asking for recommendations from friends and colleagues who use a hosting service, or checking review sites on the internet may help in finding a suitable service.

Use of a personal web server

If the researcher has a broadband-connected computer which has a permanent IP address and which is permanently connected to the internet, it may be possible to use the computer as a server on which to host the questionnaire. This will allow complete freedom over the technologies used and the server-side processing added and may prove to be a suitable solution for researchers with high-level system administration and scripting skills who wish to create a questionnaire with advanced functionalities. However, for most questionnaires, this is unlikely to be needed. Running a public website from a personal server is also a difficult task and is not recommended for those without the knowledge and experience to ensure that the server is set-up properly, has adequate disc-space to deal with the traffic received, and is secure. For a questionnaire hosted in this way, it will also be necessary to ensure that the data collected is archived regularly and held securely. If hackers find a way to access the server, not only will the questionnaire data be potentially compromised, but the server could then also be used to attack other servers over the internet.

Use of a personal web server

When choosing a hosting option, it is important to consider the URL the questionnaire will have when it is made publicly available. It may be necessary to choose and acquire a specific domain name for the questionnaire. Information about domain names is provided below.

If an institution's server is used, the URL of the questionnaire is likely to be within the file tree of the institution and/or department's website. Thus, it will have a URL which automatically includes a reference to the institution, such as the following:

```
http://www.university.ac.uk/research/questionnairename.html
```

This is likely to enhance participants' view of the legitimacy of the research.

If a commercial internet hosting service is used, many hosting services offer the possibility of having a site name as a sub site within the host's own domain name. In this case, the URL would be something like:

```
http://www.myhostingcompany.com/questionnairename/intro.html
```

In either case, if the researcher wishes to have a specific URL for the questionnaire, it will be necessary to pay for a domain name under which the questionnaire can be made available. It will also be necessary to acquire a domain name if a personal server is used so that the questionnaire can be made available online through its own URL.

Typically domain names are registered for 1-2 years for a registration fee which gives the holder sole rights to use the name. This is done through registry services, such as Nominet (<http://www.nominet.org.uk/>), which is the registry for UK domain names. The site includes a facility to search for 'unclaimed' domain names which can then be registered. Most hosting company sites also include this facility and offer to register domain names on their clients' behalf either as part of a hosting package or as a stand-alone service.

Once the domain name has been acquired, it is possible for the hosting server to be set to deliver the questionnaire in response to requests to this domain over the internet. This will make it possible for the questionnaire to have a bespoke URL, e.g:

```
http://www.researchproject.net
```

Where the researcher is affiliated to an institution but the questionnaire is not hosted on the institution's server, however, adding a link to the researcher's page on the institution's website and including a graphic of the institution logo is likely to be necessary for research legitimacy.

An informative, though perhaps not entirely impartial, introduction to hosting is available on the W3Schools website at the following address:

<http://www.w3schools.com/hosting/default.asp>

Server software

The server software handles requests for webpages from the server and also handles the interaction with server-side processing technologies and databases. In many cases, the choice of which server-side technology can be used depends on the server software that is installed. This, in turn, often depends on which operating system is in use on the computer (e.g. windows or linux). The most popular server software in use on the internet is Apache which is a free open-source option which can be used on both operating systems. Microsoft server software is also a commonly-used example, available for use on the Windows operating system and other server software is also available. Most hosts are likely to use one of these two options, though other server software is available. It is likely to be important to find out which operating system and server software are used, so that the researcher can set up a similar development environment on a personal computer to create and test the questionnaire before uploading it. Where a researcher wishes to use a particular type of software this will also be a key consideration when choosing a hosting service.

Server-side technologies

A number of technologies are available to allow server-side processing to be added to web pages. Some of the technologies are open-source and freely available for both development on a local computer and for use with servers to host public websites. The proprietary technologies are frequently also available free of charge as 'development' versions which can only be run on local machines. However, they can be expensive to buy where they are to be used with servers to host open websites. Where a researcher wishes to use this software, it will be necessary to check that it is available on the hosting server.

It is necessary to find out which operating system and server software is used by the host, and to find out which server-side processing technologies are supported. This will allow the decision on which technology to use to develop the questionnaire to be made. Where a researcher wishes to use a particular technology, it will also allow a hosting service offering this technology to be chosen. Commonly-used examples of server-side processing technologies are introduced below, and links to the official websites for each are also given.

CGI

CGI (Common Gateway Interface) is a set of rules for interaction between a server and a server-side program rather than a scripting language. Typically PERL is the language used to develop scripts for use with CGI. CGI is the oldest server-side processing technology available on the internet, and though it has largely been superceded by more efficient alternatives, it remains a popular choice. One of the key advantages of CGI is that ready-made scripts are often available online which may provide all the functionalities required to process a questionnaire without the need to develop programming skills.

<http://www.w3.org/CGI/>

PHP

A freely-available open-source server-side scripting language which can be downloaded and installed to allow the addition of server-side processing capabilities to web pages. Rather than existing separately from an HTML document, PHP scripts are embedded into web pages and are thus more efficient. PHP is designed to work simply and easily with database technologies, and it is frequently used in combination with the open-source database software MySQL. It is generally installed as an Apache module so that the Apache server software is effectively extended to include PHP functionality for the optimum performance. It can also be used with Microsoft servers.

<http://www.php.net/>

Microsoft ASP/ASP.NET

ASP is a framework developed by Microsoft for providing server-side processing capabilities and database functionalities which has largely superceded by a more recent version, ASP.NET (though ASP remains widely used at the time of writing and a wide range of resources are still available on the internet to help developers who use ASP). ASP.NET makes use of ready-made web controls to allow functionalities such as straightforward database integration or standard form validation routines to be added to web pages. The use of ASP.NET typically requires the use of a Microsoft server and operating system. Free downloads of suitable server software to allow pages to be developed and tested on a local machine are usually available, along with free development tools such as a basic version of the Visual Web Developer or the free WYSIWYG Web Matrix software.

<http://www.asp.net/>

Macromedia ColdFusion

ColdFusion allows server-side processes to be added to webpages through the inclusion of ColdFusion Markup Language, a scripting language consisting of a range of tags. These tags effectively allow common server-side tasks such as sending emails or working with databases to be created through the addition of single tags, and it is often considered to be a relatively straightforward technology to learn for those with some understanding of HTML. ColdFusion can be used on a wide range of servers, but the software is less commonly used than some of the other server-side technologies. It is proprietary to Macromedia and is only available free of charge as a development version which can only be used on a local machine or for a limited trial period in the case of the full software.

<http://www.macromedia.com/software/coldfusion/>

JavaServer Pages (JSP)

Like PHP, ASP.NET and ColdFusion, JavaServer Pages consist of code added to the HTML document. In this case, Java code is embedded in the document which is activated when the page is requested. The use of JSP depends on downloading the appropriate software, namely the Java Development Kit and a JSP/Servlet engine such as Tomcat.

<http://java.sun.com/products/jsp/>

The 'Resources' section below contains references and links to introductory information and tutorials on the use of these technologies, and to resources on the use of these technologies to perform common questionnaire processing tasks.

Database software

Where the intention is to use server-side processing technology in combination with databases it will be necessary to choose an appropriate example of database software. As with the server-side technology, a major influence on this choice is likely to be the type of database available on the hosting server, though where a researcher wishes to use a particular type of software, this will be one of the key issues in selecting a service provider.

A wide range of database software is available that can be integrated with server-side processing. The high-end databases such as Oracle, Informix or Sybase may be a suitable solution for complex research projects with adequate funds available for the purchase. However, in most cases, the two most common options are likely to be Microsoft's SQL server for use with ASP.NET, or MySQL for use with the other server-side technologies and with PHP in particular.

Setting up a development environment

Once it has been established what technologies are to be used, it is possible to download, install and configure the software needed to develop and test the questionnaire. This process will be different depending on the technologies chosen, but will typically involve the following stages:

1. Download/purchase and install the server software, following the installation instructions provided.
2. Test that the server is working correctly on the local machine. Some server software (e.g. Apache) will include a test page with the download. This will be saved in the root of the directory used by the server to hold the webpages which it will deliver. In the case of Apache, this will typically be the *htdocs* directory in the Apache program files on the hard drive (e.g. *C:\Program Files\Apache Group\Apache2\htdocs*). If no test page is available with a particular server, or if the researcher wishes to carry out a further test, an

- alternative test page (e.g. an HTML page with a name such as *'testing.html'*) can also be created and saved in this directory. The URL can then be typed into a web browser to view the page. This is typically *'http://localhost/'* when the server is running on a local machine (e.g. *'http://localhost/testing.html'*). If the installation has been successful, the test page will be displayed.
3. Download/purchase the server-side processing and database technologies to a suitable directory (e.g. if installing PHP, this should be downloaded to a file called php on the root of the hard drive).
 4. Configure the server software, the server-side processing technology and the database software to allow them to work together. This can be quite a complex process requiring settings and/or configuration files to be altered, and it is important to refer to up-to-date guidance (see below).
 5. Create a test database and a test page including a connection to this database. Save this in the root directory used by the server and access it through the browser to check that the page is displayed successfully, indicating that the software is interacting correctly (see point 2).

When carrying out this process, it is important to refer to detailed and up-to-date guidance specific to the particular technologies used. Most training books on particular software will include a step-by-step guide to installation and configuration on different platforms (see the 'Resources' section below). When choosing a book, it is important to check that this information is provided for the software versions and platform you wish to use. Guides are also available on the internet.

Using software tools

A number of tools are available to assist in the development web pages with server-side processing capabilities. The use of WYSIWYG web editors can simplify and automate many of the common tasks involved in producing a questionnaire processed server-side, such as validation routines or connecting to databases, and other tools are available to help with the creation of scripts or the administration of databases.

Macromedia Dreamweaver is a commonly-used WYSIWYG editor which offers support to users who are developing server-processed webpages. Through Dreamweaver, it is possible to set up an environment for developing pages using all of the most common server-side technologies including those mentioned above. Once this is done, tools are available within the package to allow functionalities to be automatically added such as database connections, validation or password access. If the user has a knowledge of the chosen technology and wishes to create different functionalities, tools such as tag choosers can also provide assistance with coding in the chosen language.

Microsoft FrontPage is another WYSIWYG editor which allows the development of features such as validation of form data client and server-side, sending form results to an MS Access database or connecting to an alternative external database. This can be done with little or no knowledge of server-side processing using dialogue boxes within the editor. However, taking advantage of this requires that the server the questionnaire will be hosted on allows the use of FrontPage extensions, a proprietary Microsoft technology. If you have a knowledge of the editor and a server which allows this, however, it can dramatically simplify the process and reduce the need to spend time learning about new technologies or developing programming skills.

Where ASP.NET is the chosen technology for implementing the questionnaire, development tools such as the Microsoft Visual Web Developer are available to provide a drag-and-drop interface for page creation and to simplify the creation of code and integration with databases. The free WYSIWYG Web Matrix software is also available from Microsoft for the development of these pages.

Finally, a number of tools are available to make the production of PHP code and the creation and administration of MySQL databases easier and more intuitive. Like PHP and MySQL, these

tools tend to be available open source and can be found by searching open-source repositories (See the 'Resources' section for example of these repositories).

Adding the server-side processing functionalities

Once the software and technologies used have been established, and the development environment has been set up accordingly, the pages and scripts can be developed to add server-side processing facilities to the questionnaire. In some cases, if a questionnaire has already been developed, it may be necessary to adapt the HTML to suit the server technology used. Thus, for example, if ASP.NET is used, the HTML form elements may need to be replaced with the ready-made web controls available with this technology.

Common server-side processing tasks are as follows:

- **Validation**, which should be added as a back-up to client-side validation. Facilities also need to be added to send a thank-you message on submission where no problems are found, or for the questionnaire to be redisplayed when errors are found (this should be done so that error messages highlight the problem, while the answers a participant has already entered are preserved);
- **Emailing results**, which may be the preferred method of processing the results as it is straightforward and easy to implement. The emails can also be encrypted to increase the security of sensitive data. Where a database is used, the researcher may also wish for the results to be emailed at the same time as they are added to the database.
- **Automatically adding submitted data to a database**, so that responses are added to the database on submission of the form;
- **Adding a system of password access**, so that only invited participants with access to a password can access the questionnaire and so that multiple submission using the same password is prevented;
- **Working with multi-page forms**, to pass the information gathered from page to page, or to submit each page when it is completed and then prevent participants going back and resubmitting these pages.

A wide range of books are available on the use of server-side processing technologies, along with websites offering tutorials and scripts. A number of suggestions for each of the main technologies is provided in the 'Resources' section below. This includes links to useful sources of information, tutorials and examples on how the common questionnaire processing tasks outlined above can be developed using these technologies.

Resources

General

Books

A good source of information is through the websites of key publishers in the field of web development. These include the following publishers:

O'Reilly

<http://www.oreilly.com/>

Peachpit Press

<http://www.peachpit.com/index.asp>

SAMS

<http://www.sampublishing.com/index.asp>

These sites offer facilities to search for titles related to particular technologies and also offer sample chapters and articles. They also offer access to Safari Bookshelf, which is one of the most convenient access points for books on these technologies online. It offers searchable access to the titles of these and other key publishers in the field for viewing onscreen or for downloading.

Other publishers which are not included in Safari Bookshop offer similar searchable websites and online access to their catalogues, e.g:

WROX

<http://www.wrox.com/WileyCDA/>

Apress

<http://www.apress.com/>

Websites

W3Schools

<http://www.w3schools.com/>

Provides information and tutorials on a range of server-side technologies including ASP, PHP, SQL, and ASP.NET.

Webmonkey

<http://www.webmonkey.com/>

General web-design resource. The programming section of the 'How-to library' includes tutorials on ASP, PHP, ColdFusion, and Perl/CGI.

PHP/MySQL

Books

Coggeshall, J. (2005) *PHP Unleashed*. Indianapolis. SAMS.

Kent, A., and Powers, D. (2004) *PHP Web development with Macromedia Dreamweaver MX 2004*. Berkeley, CA. Apress.

Naramore, E., Gerner, J., Le Scouarnec, Y., Stolz, J. and Glass, M. K. (2005) *Beginning PHP5, Apache, and MySQL Web Development*. Indianapolis. WROX.

Sklar, D. (2004) *Learning PHP*. Sebastapol, CA. O'Reilly.

Sklar, D. and Trachtenberg, A. (2003) *PHP Cookbook*. Sebastapol, CA. O'Reilly.

Ullman, L. (2005) *PHP and MySQL for Dynamic Web Sites*. Berkeley, CA. Peachpit Press.

Welling, L and Thomson, L. (2004) *PHP and MySQL Web Development*. Indianapolis. SAMS.

Zandstra, M. (2005) *Teach Yourself PHP in 24 Hours, 2nd Edition*. Indianapolis. SAMS.

Websites

Codewalkers

<http://codewalkers.com/>

Offers a wide range of resources on PHP and MySQL including tutorials

PHP

<http://uk.php.net/manual/en/introduction.php>

An introduction to PHP from the official website which includes a very useful introductory tutorial.

MySQL Tutorials

<http://www.php-mysql-tutorial.com/>

A series of tutorials on how to use PHP and MySQL to create and administer databases.

PHP/MySQL Tutorial

<http://www.webmonkey.com//programming/php/tutorials/tutorial4.html>

A relatively straightforward introduction to PHP and MySQL, including data validation.

ASP.NET / ASP

The resources below refer to ASP.NET which has been designed to supercede ASP. However, at the time of writing ASP remains a commonly-used server-side technology and a wide range of resources are available offering information and tutorials in its use.

Books

Duthie, G. A. and MacDonald, M. (2003) *ASP.NET in a Nutshell*. Sebastapol, CA. O'Reilly.

Hart, C., Kauffman, J., Sussman, D. and Ullman, C. (2005) *Beginning ASP.NET 2.0*. Indianapolis. WROX.

Kittel, M. A. and LeBlond, G. T. (2004) *ASP.NET Cookbook*. Sebastapol, CA. O'Reilly.

Martinez, J. and Parnell, R. (2003) *ASP.NET Development with Dreamweaver MX*. Berkeley, CA. Peachpit Press.

Mitchell, S. (2003) *Teach yourself ASP.NET*. Indianapolis. SAMS.

Walther, S. (2003) *ASP.NET. Unleashed*. Indianapolis. SAMS.

Websites

ASP.NET Quickstart Tutorial

<http://www.asp.net/QuickStart/aspnet/Default.aspx>

Detailed tutorials on using ASP.NET including information on how ASP.NET controls are used with code examples.

4 Guys from Rolla

<http://www.4guysfromrolla.com/>

Searchable resource with articles and tutorials on specific aspects of ASP.NET.

CGI/PERL

Books

Colburn, R. (2003) *Teach yourself CGI*. Indianapolis. SAMS.

Guelich, S., Gundavaram, S. and Birznieks, G. (2000) *CGI Programming with Perl*. Sebastapol, CA. O'Reilly.

Websites

CGI Programming 101

<http://www.cgi101.com/>

Tutorials aimed at beginners with information on how to set up a development environment using CGI/PERL and how to process forms and write data to files.

CGI Made Really Easy - or, Writing CGI scripts to process Web forms

<http://www.jmarshall.com/easy/cgi/>

Basic introduction to collecting and formatting information from forms.

Elated

<http://www.elated.com/tutorials/programming/perl/cgi/>

Tutorials covering a basic introduction to CGI programming with PERL along with issues such as validation and emailing.

ColdFusion

Books

Brooks-Bilson, R. (2003) *Programming ColdFusion MX*. Sebastapol, CA. O'Reilly.

Camden, R., Chalnack, L., Buraglia, A. C. and Forta, B. (2005) *Macromedia ColdFusion MX 7 Web Application Construction Kit*. Berkeley, CA. Macromedia Press.

DeHaan, J. (2004) *ColdFusion Web Development with Macromedia Dreamweaver MX 2004*. Berkeley, CA. Apress.

Mohnike, C. (2003) *Teach Yourself Macromedia ColdFusion in 21 Days*. Indianapolis. SAMS.

Websites

Macromedia's Support Centre for ColdFusion - Tutorials

http://www.macromedia.com/support/coldfusion/tutorial_index.html

A wide range of tutorials. Part of the ColdFusion Support Center which includes resources, technical notes and a forum.

EasyCFM Tutorials

<http://www.easycfm.com/tutorials/index.cfm?dirView=True>

A comprehensive range of ColdFusion tutorials.

JSP

Books

Bergsten, H. (2003) *JavaServer Pages, Third Edition*. Sebastapol, CA. O'Reilly.

Brunner, R. (2003) *JSP: A Practical Guide for Programmers*. San Fransisco, CA. Morgan Kaufmann Publishers.

Holzner, S. (2002) *Teach Yourself JavaServer Pages in 21 Days*. Indianapolis. SAMS.

Perry, B. W. (2004) *Java Servlet & JSP Cookbook*. Sebastapol, CA. O'Reilly.

Websites

Caucho JSP Tutorials

<http://www.caucho.com/resin-3.0/jsp/tutorial/index.xtp>

Covers topics including form processing and emailing form contents.

JSP Tutorial

<http://www.jsptut.com/>

Series of tutorials covering the basics of JSP and dealing with forms processing, databases and emailing. States that users should have a knowledge of HTML and Java.

JSP Olympus

<http://www.jspolympus.com/JSP/JSP.jsp>

Comprehensive range of tutorials.

Frequently-asked questions

I don't know anything about computers. Will I be able to set up an online questionnaire?

The simple answer is yes. There are a number of ways that you can set up a questionnaire. You do not need to be an expert in computers, but you may need to be prepared to spend some time learning some new skills.

If you do not want to spend a great deal of time doing this, producing the questionnaire yourself is not likely to be the best option. If you are working in an institution, it is a good idea to check what support is available to you. It may be that there are systems in place which you can use to get a questionnaire online.

Alternatively you could use one of the many online questionnaire software and hosting services that are available. These aim to make it possible to get a questionnaire online and gather results with little or no technical skills. They generally use a forms-based interface to take you through the whole process of developing and implementing the questionnaire. There are a wide range of options available for different budgets and with different features, and you may need to spend some time working out which of the services to choose. The 'Choosing software' section includes an activity designed to help you to work out what features you will need for your questionnaire. It will allow you to develop a checklist of features that you can use when comparing products. The 'Using software' section also provides an outline of the typical procedure for creating and administering a questionnaire using these products.

The introduction to this 'Technical guide' module provides a detailed overview of the different methods you can use to implement a questionnaire and the technical skills and knowledge that will be required depending on the method you choose.

How long will it take me to learn to set up an online questionnaire?

From a technical perspective, this very much depends on what your experience of web design is at the outset, what support is available, and what method you choose to get the questionnaire implemented.

If you are developing a questionnaire yourself and you have little experience of web design, you will need to spend some time learning about technologies such as HTML and CSS in order to set up a web form and related pages such as an informed consent page. You can also use WYSIWYG (What You See Is What You Get) software such as Macromedia Dreamweaver or Microsoft FrontPage to develop the questionnaire, although an understanding of HTML will help you with this. You should set aside around 10-20 hours to learn the skills to develop the questionnaire and ensure it is designed effectively. If you plan to add extra features to your questionnaire such as validation or the gathering of information about participants' computers, you should expect to spend at least this time again to learn how this is done. Finally, if you do not have support to help you to upload the pages to a server and add server-side processes to validate and deal with the data collected, this is likely to take another 15-20 hours.

Of course any institutional support that may be available (such as an automated service to email results to the researcher) will reduce this time, but in this case you are likely to need to spend a couple of hours working through the details of the system to ensure that the questionnaire you create is suitable and meets any conditions that may be imposed.

If you use an 'off-the-shelf' software and hosting service you should expect to spend an hour or so finding the right service for your needs and budget and then to spend some time working

out how the software is used to develop the questionnaire and how to use the different options available. You should have a fully working questionnaire in a few hours.

What equipment will I need?

The minimum equipment you will need to design the web pages for the questionnaire is a simple text editor, such as notepad for windows, and a browser such as MS Explorer in which you can test your pages. WYSIWYG (What You See Is What You Get) software such as Macromedia Dreamweaver or Microsoft FrontPage can also be used to aid the design of the questionnaire.

You will also need access to a server on which you can upload your questionnaire to make it available online. Depending on how you choose to process the data, you may also need access to a database on the server (Alternatively, you may choose to email the results). In most cases, a simple text editor is also the only thing required for the creation of server-side processes to deal with the questionnaire data. Depending on the server and the server-side technologies available to you, however, you may also choose to download software which will help you to do this. See the 'Server-side processing' section for more information about the different technologies and software that can be used.

Will I be able to get help from my University/organisation?

If you are working within an institution such as a university, a government department, an NHS trust, or a charitable organisation, technical support may be available which may even extend to the bespoke conversion of a paper-based questionnaire to a web version. If this is not available, there may also be standard procedures in place for implementing web forms once they have been created by the researcher. This may, for example, involve a mailing facility whereby the data is automatically formatted and delivered by email when the form is submitted. It may also involve a facility allowing the data to be downloaded in an appropriate format for importing into a statistical analysis, database or spreadsheet package. The institution may also have purchased a site licence for an off-the shelf survey creation and administration package which may be suitable for the purposes of the research project, and in educational institutions in particular, site licences may also have been purchased for assessment software which may provide the facilities required for the implementation of a basic questionnaire.

The first step in finding out about these systems is usually to check the institution intranet or to contact computer services.

Are there any limits to the number of question types I can have?

In HTML pages, there are five basic forms of form controls for inputting data as follows:

<p>Text box</p> <input type="text"/>
<p>Check boxes</p> <input type="checkbox"/> Yes

No
 Maybe

Radio buttons

Yes
 No
 Maybe

Text area

Insert text here

Select box

Choose an option

Of course, these can be organised in different ways to create different question types. For example, tables can be used to group radio buttons into grids for Likert scales or semantic differential questions, as in the following example:

Complete the following statement by choosing the number that most closely matches your opinion for each row:

The internet is:

	1	2	3	4	5	
boring	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	interesting
difficult	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	easy
risky	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	safe
useless	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	useful

See the 'Web forms' section of this 'Technical Guide' module for further details.

What is the best question type?

There is no simple answer to this question as this is likely to depend on the context of the research. An activity designed to explore the length of time needed to complete different types of questions is available in the 'Design issues 2: Content' section of the questionnaires module.

Does it matter what I put in as the default answer?

For check boxes, radio buttons and select boxes, it is possible to specify which of the options is specified by default as shown below:



The screenshot shows a questionnaire form with three sections:

- Check boxes:** Three options are listed: 'Yes' (unchecked), 'No' (unchecked), and 'Maybe' (checked).
- Radio buttons:** Three options are listed: 'Yes' (selected), 'No' (unselected), and 'Maybe' (unselected).
- Select box:** A dropdown menu with the text 'Choose an option' and a downward arrow.

In the case of radio buttons and check boxes it is a good idea not to set a default option as it will be impossible to establish whether or not the option was actively selected by the participant or whether the question was unanswered. For select boxes, if the default answer is 'Choose an option' as in the example above, this will not be a problem this answer will clearly indicate non-response.

Is it better to have a single-paged questionnaire or one that spans multiple pages?

One-page questionnaires are generally easier to implement as they consist of a single web form which can easily be processed through the submission of this one form. Where the questionnaire is relatively short and straightforward in terms of structure, this is likely to be the best option.

However, with longer or more complex forms, attempting to present the entire questionnaire in one page may lead to problems such as the following:

1. Presenting all questions at the same time may give an impression of greater length which may discourage participants from proceeding.
2. Opportunities to validate individual questions or smaller groups of questions as the participant progresses through the questionnaire may be reduced (see the 'Form validation' section). In turn, this may lead to frustration if all questions are validated at once at the end of the questionnaire.

3. Although skip patterns can be introduced through linking to anchors further down a page or through instructing participants to skip a question by scrolling to the next, this may not be the most effective or intuitive method of delivering the questions.
4. If a participant drops out mid-questionnaire, all data will be lost and there will be no opportunity for collection of partially-completed questionnaires or for identification of questions that may be precipitating drop out.

For longer questionnaires, the use of multiple pages can add to the effectiveness of question delivery, providing clearer routes through the questions and offering the opportunity for a more sophisticated presentation of skip patterns. For example, links to different sections can be added which participants can be prompted to select according to the answer to key questions. However, because all the questions are not made visible, submitted and processed at the same time, a number of extra aspects must be considered

1. An indication of progress through the questionnaire must be given, either through the use of a progress bar (see 'Key design issues' section of this guide) or through structuring the questionnaire into different sections and indicating the nature of this structure to respondents (see 'design issues 1' in the questionnaires modules). If this is not done effectively, uncertainty over progress or a realisation that the indicators of progress are inaccurate, may lead to frustration and drop out.
2. A decision must be made on whether data should be submitted for processing at the end of each page, or at the end of the questionnaire. If it is done at the end of each page, this will allow partially-completed questionnaires to be collected and any problem questions to be identified, but measures must be taken to identify or prevent multiple submission of any sections. If it is done at the end of the questionnaire, it will be necessary to pass information supplied up to a given point in the questionnaire from page to page.
3. As participants progress through the questionnaire, they may wish to return to a previous page to review and change answers. Unless measures are taken to ensure that the data they have already entered is still available when they do this, it is important to inform them that answers already entered may be no longer available if they go back. It may also be important to add instructions not to go back through a questionnaire if each page is to be submitted individually, or to add validation routines preventing submission a second time.

There is a good chance that my survey group will be using old computers. Is there anything I should consider?

There are a number of measures that can be taken to ensure that a questionnaire is suitable for older browsers and equipment. These include maintaining a straightforward design, avoiding the use of third-party plug-ins such as Macromedia Flash, ensuring it is designed for accessibility (see question below) and validating the HTML and or CSS used to create it. However differences in aspects such as the size and appearance of form elements and tables may remain when viewed on different systems.

It is thus good practice to test pages on as many different browsers and systems as possible, and as an absolute minimum to install the latest versions of the three most popular browsers on your desktop and use these to test. Friends and acquaintances with different systems (e.g. AppleMacs or PCs) and older versions of browsers can also be called on to test for any problems. Where design problems are found in particular systems and browsers, an attempt can then be made to change the design to best accommodate them.

It is also possible to collect information about the user's computer and browser alongside the data from the questionnaire (see the 'Gathering information about participants' section of this technical guide). This can allow an overview of the technologies available to respondents to be

gained. If it becomes clear, for example, that older or less common browsers are being used to access the questionnaire, it can be tested on these browsers and redesigned if necessary.

It is also a good idea to test the questionnaire on different screen size settings to ensure that questions are visible in their entirety and do not require scrolling, to use web-safe colours which will display consistently on different monitors and to use common fonts and 'font-families' to ensure that the fonts used will display consistently on different systems.

See 'consistency' part of the 'Key design issues' section of this guide for further information.

How can I make sure my questionnaire can be used by participants with disabilities?

It is important to design pages to be as accessible as possible and there are a number of simple steps that can be taken to increase the accessibility of an online questionnaire and its associated web pages. These can ensure that the contents are accessible to users with a range of user-agents including text-only and screen reading browsers and other assistive technologies.

Designing the site to be compliant with standards set out by the World Wide Web Consortium (W3C) is an important step in ensuring accessibility. The 'Resources' section of this guide includes a link to W3C's validation tools which allow web pages to be checked for standards compliance. By uploading a page or entering the URL, the tools will run automated tests and report on any pieces of invalid markup in the pages.

It is also good practice to separate content from presentation in web pages by using Cascading Style Sheets (CSS) to add design features (See the 'Introduction to CSS' section of this guide). Although it should be remembered that this may lead to increased inconsistencies in display on older browsers, this allows participants to control how the site should be presented. They can override style information to allow presentational features such as text size, font, colour and layout to be changed according to need.

Beyond this, the W3C Web Accessibility Initiative guidelines (WAI) includes a wide range of measures which should be taken to ensure that a website is accessible (<http://www.w3.org/WAI/intro/wcag.php>). The guidelines divide these measures into Priority 1, 2 or 3 according to how essential they are to accessibility. Some key measures that should be taken for accessibility are shown in the 'Key design issues' section of this guide.

Can I create something that people can fill in on TVs, mobile phones?

A well-designed questionnaire should be usable on a wide range of web-enabled devices including TVs, mobile phones and Personal Digital Assistants (PDAs). By designing for accessibility and separating content from presentation (see question above), you will be able to make sure that the pages are as accessible as possible to devices such as these as well as to text-only and screen-reading browsers and other assistive technologies.

Glossary

A

Accessibility

The accessibility of a web page refers to the extent to which users can access the content regardless of the technology they use and any disability they may have. An accessible web page is one that is designed to ensure that this is possible through, for example, providing textual descriptions of graphics used to allow their significance to be described in text-only browsers or via screen-reading software.

Active server pages (ASP)

A framework developed by Microsoft for providing server-side processing capabilities and database functionalities. Now largely superseded by a more recent version, ASP.NET.

Apache

Freely-available open-source server software which is one of the most widely-used examples of server software on the internet.

B

Body

The body section of an HTML document contains the main display content. It is here that text, images, links, form elements, tables and lists are placed.

Browser

Software which requests resources (mainly web pages) from a server computer and displays them. An example of client software held on a client machine.

Button

A standard HTML button. Can be linked to JavaScript and perform an action when clicked.

e.g.:

HTML: `<input type="button" value = "Standard button" />`

C

Check box

Square tags that display a mark when selected and can allow multiple responses.

e.g. A B

HTML: `<input type="Check box" name="1" value="A"> A<input type="Check box" name="2" value="B"> B`

Client

The software that allows a computer to request web pages from a server computer and displays these pages. Also used to refer to the computer on which this software is held.

Client-side scripting

Client-side scripting through scripting languages such as JavaScript allows dynamic or interactive features to be added to web pages. Code is added to a web page which is executed in the browser on the client computer. It can be used to, for example, carry out different actions according to user actions or input.

CSS

CSS (Cascading Style Sheets - also referred to simply as 'Style Sheets') provide a means of adding design elements to basic HTML pages. For example, using CSS, it is possible to control the colour, positioning and spacing of objects such as text, links, images and tables.

D

DOCTYPE

A DOCTYPE (document type) definition, known as a DTD should begin an HTML document. This declares what type of page it is and what language is being used, and it allows the page to be validated as conforming to Worldwide Web Consortium (W3C) standards.

E

Event handler

An event handler is a piece of code in programming or scripting languages such as JavaScript that triggers an action when a particular event occurs. Examples of event handles are those that trigger actions when the mouse is clicked, double-clicked or moved, a key is pressed, or a page is loaded or unloaded.

F

Form elements/controls

A set of form items that the user can enter data into to be sent to the researcher. Some commonly-used elements are as follows:

Button, Check box, Select box / drop down list box, Password box, Radio buttons, Text area, Text box.

Form tags

Form tags mark off the beginning and end of a form. Controls within the form tags are effectively grouped together so that when a submit button is clicked the data in all the controls within the form is sent for processing. It is possible to include multiple forms on a page, but only one form can be submitted at any one time.

Function

A function is a block of code in programming or scripting languages such as JavaScript that carries out a particular action. In effect the code is not carried out until the function is 'called' from within the document when a particular event occurs (such as the user clicking a submit button).

G

Graphic Interchange Format (GIF)

One of the two most common types of images in use on the internet (along with JPEGs), GIFs are usually more appropriate for line drawings or graphics with a limited number of colours.

H

Head

The head section of an HTML document contains information which is basically not intended for display. It is loaded into the browser before the body section and typically includes the DOCTYPE, the page title, meta commands and any CSS and client-side script information.

Hidden form fields

Hidden form fields are form controls that are not displayed on the page (though they are visible in the HTML source for the page). They are useful for storing and passing information from page to page which is not necessary or desirable to display. They can be thought of as text boxes with content that can be set by the developer via HTML or JavaScript rather than being completed by the user.

HTML

HTML (Hyper Text Markup Language) is the technical language that lies behind most web pages. It consists of tags which surround blocks of text to indicate how they should appear in a browser, and which are used to insert elements such as images or tables.

HTML tags

In an HTML document, tags are used to tell the browser how to present the layout and style of text and other elements. Tags can consist of elements, attributes and values. The element indicates what should be displayed in the browser, and the attributes and values indicate how this should be displayed.

e.g. In the following tag, `<p align="center">Hello!</p>`, `<p></p>` is the element which tells the browser to display a new paragraph, **align** is the attribute and **center** is the value which indicates that it should be displayed with a centered alignment.

I

Internet Protocol (IP) Address

A string of four numbers separated by full-stops which provide a unique identifier for all computers permanently connected to the internet.

J

Joint Photographic Experts Group (JPEG)

One of the two most common types of images in use on the internet (along with GIFs), JPEGs are usually suitable for images with a large number of colours such as photographs. The file extension is '.jpg'.

JavaScript

The most popular client-side scripting language in use on the internet. JavaScript code is added to an HTML document and is executed in the browser on the client computer. It can be used to, for example, carry out different actions according to user actions or input.

K

L

M

MySQL

A freely-available open source database server, which can be downloaded and installed to allow database functionality to be added to a web page. It is frequently used in combination with PHP to allow databases to be added to and accessed over the internet.

N

O

Open source

In general terms, open source software allows for users to access the source code for free and allows it to be modified and redistributed. A full definition is available at:
<http://www.opensource.org/>

Optimisation

The process of reducing as much as possible the file size and download time of resources such as web-graphics while maintaining a suitable level of quality.

P

Password box

Text input box that allows a single line of text to be entered. It is possible to limit the size and number of characters that can be entered. As the user types, the characters are hidden from display.

e.g.

HTML: `<input type="password" size="15" maxlength="10" />`

Path

A reference to a file and it's location in a series of folders held on a computer.

e.g. **C:\Documents and Settings\My Documents\main_site\section1\page1.htm** refers to an HTML page called 'page 1' held in the c drive of a local computer in a folder called section 1, held in a series of folders in 'Documents and Settings'.

PHP

A freely-available server-side scripting language which can be downloaded and installed to allow the addition of server-side processing capabilities to web pages.

Q

R

Radio buttons

Circular tags that fill in when one option is selected.

e.g. Yes No

HTML: `<input type="radio" name="1" value="Yes" / > Yes <input type="radio" name="1" value="No" /> No`

Reset button

A reset button clears any form data that has been input, returning them to the original values they had when the page was loaded.


e.g.: 

HTML: `<input type="reset" value="Reset" />`

S

Select box / Drop-down list box

An element which allows users to select options by clicking. Only one option is displayed until the user clicks on the arrow.

e.g. 

HTML: `<select name="select"> <option>Option 1</option>
<option>Option 2</option> <option>Option 3</option>
<option>Option 4</option> </select>`

Server

A computer which delivers web pages to a client computer when a URL is typed into the address bar of a browser on that computer. Also used to refer to the software held on the server computer which allows this process to take place.

Server-side processing


Server-side processing allows dynamic or interactive features to be added to web pages. This is done by the server computer before the page is sent to the client computer. Server-side processing can be accomplished using a range of technologies such as PHP, ASP(X), Perl/CGI and ColdFusion. It can be used to, for example, validate and process information entered by users into web forms, store or retrieve information in databases, and automatically send emails.

Skip mechanisms

Functionalities added to an online questionnaire which automatically provide participants with a route through the questionnaire, avoiding questions that are not relevant. When a question is answered, the next question will be delivered according to the response so that different questions are provided depending on particular answers.

Submit button

A submit button sends the form data to the server when clicked. The action of doing this depends on the form action specified. Most commonly it will be to email the results or add them to a database.

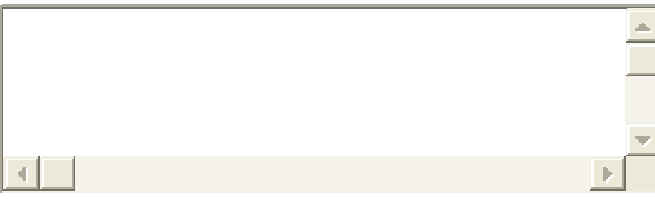
e.g.: 

HTML: `<input type="submit" value="Submit" />`

T

Text area

Allows the user to input a large amount of text. By default, the text will wrap onto a new line when the end of a line is reached, and a scroll-bar will appear on the right-hand side when the number of lines displayed is exceeded.

e.g. 

HTML: `<textarea cols="60" rows="5"></textarea>`

Text box

Allows a single line of text to be input of a size and number of characters specified.

e.g. 

HTML: `<input type="text" size="25" maxlength="20" />`

Text editor

A simple application which allows users to enter, edit and save text, typically with basic formatting options.

U

URL

A URL (Uniform Resource Locator) is the address for a resource available online (usually a web page). URLs consist of a reference to the server computer which holds the resource along with a path to the file containing this resource on the computer. By typing the URL into a browser, a request is sent from a user's computer (client computer) to the server computer to deliver this resource.

V

Validation

The functionality which allows forms to be automatically checked before or at submission to ensure that any required questions have been answered and/or that data has been entered in a suitable format. This can be done using client-side and/or server-side scripting. Typically, validation routines will prevent submission where problems are found with the form and a message will be delivered to the user prompting them to check their answers and resubmit.

W

'Web-safe' colour palette

A set of 216 colours recommended for use on the internet as they are not subject to variation on different types of monitors and systems.

WYSIWYG

WYSIWYG (What You See Is What You Get) software packages such as Macromedia Dreamweaver or Microsoft FrontPage are tools that allow web pages to be created and edited using an interface that displays the page as it will appear in a browser.

X

Y

Z

Resources

Listings of software and services for online questionnaire production

Evans, J. R. and Mathur, A. (2005) The value of online surveys, *Internet Research*, 15, 2, 195- 219.

An examination of the involvement in online surveys of the largest US-based and global market research firms. Provides an extensive list of the services offered as of late 2004.

The WebSM searchable database

<http://www.websm.org/index.php?fl=0&p1=1123&p2=82&p3=1086&id=1086>

Database of online questionnaire software and services which has entries categorised by type, code, cost, language and country.

The Association for Survey Computing searchable software register

<http://www.asc.org.uk/Register/index.htm>

Searchable register which includes information about the software it contains such as listings of the features offered and suppliers.

Web-based survey software

<http://www.web-based-surveys.com/>

Directory of software which can be browsed or accessed using the 'software finder' which allows users to specify the features they require.

Services offering software plus hosting

The following examples are chosen as being representative of some of the different types of services available as of December 2005. In each case, a range of comparable options may be available.

Bristol Online Surveys

<http://www.survey.bris.ac.uk/>

Targeted at institutions requiring the option to have a number of different surveys and survey administrators. Highly customisable to the style needs of institutions including an option to run surveys on their server with an address that appears to be that of the institution.

Educara Survey

<http://www.educara.com/educara.cgi/survey.html>

An open source tool which offers hosting for an annual fee. The price rises according to the type of use, with students paying the least, and commercial organisations the most.

Research together

<http://www.doctoralstudents.com>

Targeted at research students. Offers a simple questionnaire production and hosting service, allowing users to download results as comma-separated values for import into an analysis package. Relatively inexpensive one-off payment, but without many of the more sophisticated functionalities such as email list management or analysis tools.

SurveyConsole/QuestionPro

<http://www.surveyconsole.com/> / <http://www.questionpro.com/>

Both are divisions of the surveyanalytics company and they use the same software and interface, but with different pricing. May offer sponsored use for academic or not-for-profit projects if certain conditions are met. Also offer a range of free resources such as articles and question templates.

Surveymonkey

<http://www.surveymonkey.com/>

Compares well with many of the other available services in terms of features, but is one of the cheapest commercial options.

Surveywriter

<http://www.surveywriter.com/site/>

Relatively expensive, but unusual in that charges are not made per period of use, but per completed survey and email invitation with a minimum of 200.

Surveyz!

<http://www.surveyz.com/>

A range of relatively sophisticated features. targeted at individual researchers or at institutions. Offers academic pricing and free use for academic projects if certain conditions are met. Also has a range of resources such as articles on online questionnaires and copyable templates of questionnaires and questions.

Websurveyor

<http://www.websurveyor.com/>

Relatively expensive, but with a wide range of features. Offers both hosting and software only solutions.

Zoomerang

<http://info.zoomerang.com/>

Offers pricing for not-for-profit and educational institutions. Also offers a range of research services such as questionnaire administration, panel services and translation.

Commercial questionnaire software

The following are some examples of commercial software for online questionnaires. A range of other options are available.

Questionmark Perception

<http://www.questionmark.com/uk/home.htm>

An educational assessment tool which offers many of the key features needed to create basic online questionnaires and has some of the more advanced features such as randomisation of questions and conditional branching. Potentially useful option if the software is available through the researcher's institution.

SelectSurveyASP

<http://www.classapps.com>

Relatively inexpensive. Offers 'classic' and 'advanced' versions with different levels of features at different prices. Has a working online demo and a useful example survey which includes comments on the features illustrated by particular questions. Offers a free installation service and free technical support.

Snap Surveys

<http://www.snapsurveys.com/>

Extensive options for mixed-mode surveys, offering a 'core product', Snap Professional, with add-ons for questionnaires via internet and PDAs, and to allow scanning and multiple data entry. Expensive example of 'high-end' options.

SphinxSurvey

[http://www.sphinxdevelopment.co.uk/ Products_sphinx.htm](http://www.sphinxdevelopment.co.uk/Products_sphinx.htm)

Extensive analysis tools including a version offering lexical analysis. Educational and public-sector pricing offered.

StatPac

<http://www.statpac.com/>

The online questionnaire software does not include analysis tools, but it can be purchased

alongside the statistics tools offered. Has basic statistical tools or an advanced version allowing multivariate statistical techniques. Technical support and updates are available free for three months, but are chargeable via annual support/maintenance agreements thereafter. A fully-functional version of the software can be downloaded and used for free, limited to 35 respondents for each survey. Download includes tutorials and extensive user guide.

Open-source software

General information

SourceForge.net

<http://sourceforge.net>

Searchable repository of open source projects.

Freshmeat.net

<http://freshmeat.net/>

Listing of new software releases.

OSS Watch: Top Tips For Selecting Open Source Software

<http://www.oss-watch.ac.uk/resources/tips.xml>

Page from the website of OSS, a JISC-funded open source advisory service. Offers guidance on selecting open source software.

Examples of questionnaire software

The following are some of the main examples of open source software for online questionnaires (generally the more established and/or sophisticated options). A range of other options may be available.

Educara Survey

<http://www.educara.com/educara.cgi/survey.html>

In addition to the software, offers hosting for an annual fee. The price rises according to the type of use, with students paying the least, and commercial organisations the most.

LE Survey

<http://le-survey.sourceforge.net/>

Designed as a tool for running questionnaires as part of longitudinal studies. Allows respondents' responses to be matched to responses to previous questionnaires while maintaining confidentiality. In early stages of development at the time of writing.

phpESP

<http://phpesp.sourceforge.net/>

Well-established software with a working demo available allowing the features and user-interface to be tested.

phpSurveyor

<http://www.phpsurveyor.org/>

A range of relatively sophisticated features. Well-established with useful documentation. Working demos are available allowing the features and user-interface to be tested.

Mod_Survey

<http://gathering.itm.mh.se/modsurvey/>

Well-established with sophisticated features such as dynamic content generation depending on previous answers. Requires the user to learn to use XML syntax particular to the software.

VTSurvey

<http://vtsurvey.sourceforge.net/>

Easy to use and particularly useful for straightforward questionnaires as only the four main

types of questions are supported (Multiple choice with radio buttons and Check boxes, and short and long text entry boxes).

HTML

Getting started with HTML

<http://www.w3.org/MarkUp/Guide/>.

A good basic introduction to HTML from the World Wide Web Consortium. (W3C).

HTML Goodies

<http://www.htmlgoodies.com/>.

A range of short tutorials designed to help you with specific aspects of web design.

HTML Reference

http://www.w3schools.com/tags/ref_byfunc.asp.

List of HTML tags organised by their function, from W3Schools.

World Wide Web Consortium (W3C) HTML Validator

<http://validator.w3.org/>.

Enter a link to your web pages or upload a local file to check that your HTML meets web standards and guidelines.

HTML Tidy

<http://www.w3.org/People/Raggett/tidy/>.

Automatically cleans up HTML to correct any problems caused either by mistakes or automatic production of invalid HTML by web editors.

WC3 Links Checker

<http://validator.w3.org/checklink>.

Automatically checks for broken links in an HTML document.

CSS

Cascading Style Sheets (CSS)

<http://www.w3.org/MarkUp/Guide/Style.html>.

An introduction to Cascading Style Sheets from W3C.

CSS Reference

http://www.w3schools.com/css/css_reference.asp.

A reference to the properties and possible values that can be applied to different elements of an HTML page from W3Schools. Also offers further information on the use of different properties.

W3Schools CSS Tutorial

<http://www.w3schools.com/css/default.asp>

Tutorials which include examples and quizzes.

The Worldwide Web Consortium (W3C)'s CSS page

<http://www.w3.org/Style/CSS/>

Offers a wealth of information on CSS.

W3C CSS Validator

<http://jigsaw.w3.org/css-validator/>.

Makes it possible to check that your CSS meets web standards and guidelines by entering a link to your CSS file, uploading your file from your computer, or pasting your CSS into a text box on the page.

Writing Efficient CSS

<http://www.communitymx.com/content/article.cfm?cid=90F55>

A useful article by John Gallant and Holly Bergevin on using CSS 'short hand' properties to reduce the size of CSS files and increase efficiency.

JavaScript

JavaScript Primers

<http://www.htmlgoodies.com/primers/jsp/>.
30 short JavaScript lessons with learning activities.

JavaScript examples

<http://JavaScript.internet.com/>.
Over two-thousand examples of JavaScripts organised into sub-sections.

JavaScript use in forms

<http://www.irt.org/script/form.htm>.
A range of JavaScript examples with source code specifically related to web-forms.

Accessibility

W3C Web Accessibility Initiative guidelines (WAI)

<http://www.w3.org/WAI/intro/wcag.php>
Document which explains how to make Web content accessible to people with disabilities. Includes checkpoints of actions that will improve accessibility and gives each checkpoint a priority rating according to its importance. Also has links to explanations of how the checkpoints can be achieved with HTML and CSS.

Accessibility validator

<http://bobby.watchfire.com/bobby/html/en/index.jsp>.
By entering the link to your page, you will receive a report highlighting any potential accessibility problems.

Vischeck

<http://www.vischeck.com/vischeck/>
A service which simulates the appearance of pages to users with different forms of colour blindness to allow pages to be tested for suitability for colour-blind users.

DEMOS Project - Guide to accessible web pages: User control

<http://jarmin.com/demos/access/control.html>
A guide to making changes to the display of web pages in different browsers.

TechDis

<http://www.techdis.ac.uk/>.
A Joint Information Systems Committee (JISC)-funded advisory service on accessibility issues in education. Contains useful resources and 'how to' guides on accessibility.

Macromedia's Accessibility Resource Center

<http://www.macromedia.com/resources/accessibility/>.
Offers general guides to accessibility and accessible design as well as specific guidance on how to increase the accessibility of HTML and multi-media produced with Macromedia products such as Dreamweaver and Flash.

Design issues

Usability news: Wichita State University Software Usability Research Lab

http://psychology.wichita.edu/surl/usability_news.html.
Newsletter providing a range of articles with information on research into software and website design and usability.

Usable Information Technology

<http://www.useit.com/>.

Leading site on usability and user studies by Jacob Neilson.

Sit Back and Relax: A Guide to Producing Readable, Accessible Onscreen Text

<http://readability.tees.ac.uk/>.

Guide to formatting text documents, especially long text documents, so that they may be easily read from computer screens without the need for printing. Produced by Bruce Ingraham and Emma Bradburn of the University of Teesside. Includes a range of Cascading Style Sheets designed for readability.

Web-safe colours

<http://www.lynda.com/hex.html>.

Offers tables of web-safe colours organised by either hue (colour) or value (lightness). Makes it easier to design appropriate colour schemes.

W3C Browser information

<http://www.w3schools.com/browsers/default.asp>

Information on the main browsers available at the time of writing is available on the Worldwide Web Consortium (W3C) website along with statistics on the usage levels of each (though it should be noted that the statistics are based on users of the site and should thus be generally considered to be skewed in favour of more technically proficient users).

MOOCK Flash Player Inspector

http://moock.org/web_design/flash/detection/moockfpi/

Provides a good explanation of Flash detection (and its limitations) along with free scripts to carry it out.

Form validation

Client-side validation using JavaScript

Form Validation Using Javascript

<http://www.4guysfromrolla.com/webtech/091998-1.shtml>

An example of form validation using JavaScript with explanations from the 4guysfromrolla.com site.

Javascript form validation – doing it right

<http://www.xs4all.nl/~sbpoley/webmatters/formval.html>

Interesting discussion of validation using JavaScript aiming to point out some of the main pitfalls.

Server-side validation

Depending on the technology used, the following links may provide a useful source of information, tutorials and examples on how to add server-side validation.

PHP/MySQL

WebMonkey PHP/MySQL Tutorials

http://webmonkey.wired.com/webmonkey/99/21/index4a_page2.html?tw=programming

A relatively straightforward introduction to validation using PHP.

Common Form Validations

<http://codewalkers.com/tutorials/47/2.html>

A PHP tutorial which covers a range of different validation activities, leading to an example of a complete form validated via PHP.

Books

Coggeshall, J. (2005) *PHP Unleashed*. Indianapolis. SAMS.
Chapter 4: Working with Forms in PHP.
Chapter 5: Advanced Form Techniques.
<http://www.sampublishing.com/ title/067232511X>

Sklar, C. (2004) *Learning PHP*. Sebastapol, CA. O'Reilly.
Chapter 6: Making web forms.
<http://www.oreilly.com/ catalog/learnphp5/>

Sklar, D. and Trachtenberg, A. (2003) *PHP Cookbook*. Sebastapol, CA. O'Reilly.
Chapter 9: Forms.
<http://www.oreilly.com/ catalog/phpckbk/>

Zandstra, M. (2005) *Teach Yourself PHP in 24 Hours, 2nd Edition*. Indianapolis. SAMS.
Chapter 9: Working with Forms (available as a sample chapter)
<http://www.sampublishing.com/ title/0672323117>

ASP.NET

Microsoft's ASP.NET framework offers a number of ready-made web controls designed to carry out server-side and, where available, client-side validation of web forms. These include controls that check required fields have been completed, that check that data in particular ranges or patterns has been entered (e.g. in the format of a telephone or credit-card number), and that compare data from one form element for consistency with that from another.

ASP.NET Quickstart tutorials: Validating Form Input Controls

<http://beta.asp.net/ QuickStartv20/ aspnet/doc/ validation/ default.aspx>
A useful tutorial on how these controls work with code examples.

Form Validation with ASP.NET - It Doesn't Get Any Easier!

<http://www.4guysfromrolla.com/ webtech/090200-1.shtml>
Another step-by-step guide.

Books

Mitchell, S. (2003) *Teach yourself ASP.NET*. Indianapolis. SAMS.
Chapter 12. Validating User Input with Validation Controls.
<http://www.sampublishing.com/ title/0672325438#>

Walther, S. (2003) *ASP.NET. Unleashed*. Indianapolis. SAMS.
Chapter 2: Building Forms with Web Server Controls.
Chapter 3: Performing Form Validation with Validation Controls.
<http://www.sampublishing.com/ title/0672325438#>

PERL/CGI

Form Validation with Perl/CGI

http://www.elated.com/ tutorials/ programming/ perl_cgi/ form_validation/
An introductory tutorial for adding validation using PERL/CGI.

Books

Colburn, R. (2003) *Teach yourself CGI*. Indianapolis. SAMS.
Chapter 7: Validating user input.
<http://www.sampublishing.com/ title/0672324040>

Guelich, S., Gundavaram, S. and Birznieks, G. (2000) *CGI Programming with Perl*. Sebastapol, CA. O'Reilly.
Chapter 4: Forms and CGI;

Chapter 8: Security (available as a sample chapter).
<http://www.oreilly.com/catalog/cgi2/toc.html>

Server-side processing technologies

General

Books

A good source of information is through the websites of key publishers in the field of web development. These include the following publishers:

O'Reilly

<http://www.oreilly.com/>

Peachpit Press

<http://www.peachpit.com/index.asp>

SAMS

<http://www.sampublishing.com/index.asp>

These sites offer facilities to search for titles related to particular technologies and also offer sample chapters and articles. They also offer access to Safari Bookshelf, which is one of the most convenient access points for books on these technologies online. It offers searchable access to the titles of these and other key publishers in the field for viewing onscreen or for downloading.

Other publishers which are not included in Safari Bookshop offer similar searchable websites and online access to their catalogues, e.g:

WROX

<http://www.wrox.com/WileyCDA/>

Apress

<http://www.apress.com/>

Websites

W3Schools

<http://www.w3schools.com/>

Provides information and tutorials on a range of server-side technologies including ASP, PHP, SQL, and ASP.NET.

Webmonkey

<http://www.webmonkey.com/>

General web-design resource. The programming section of the 'How-to library' includes tutorials on ASP, PHP, ColdFusion, and Perl/CGI.

PHP/MySQL

Books

Coggeshall, J. (2005) *PHP Unleashed*. Indianapolis. SAMS.

Kent, A. and Powers, D. (2004) *PHP Web development with Macromedia Dreamweaver MX 2004*. Berkeley, CA. Apress.

Naramore, E., Gerner, J., Le Scouarnec, Y., Stolz, J. and Glass, M. K. (2005) *Beginning PHP5, Apache, and MySQL Web Development*. Indianapolis. WROX.

Sklar, D. (2004) *Learning PHP*. Sebastapol, CA. O'Reilly.

Sklar, D. and Trachtenberg, A. (2003) *PHP Cookbook*. Sebastapol, CA. O'Reilly.

Ullman, L. (2005) *PHP and MySQL for Dynamic Web Sites*. Berkeley, CA. Peachpit Press.

Welling, L. and Thomson, L. (2004) *PHP and MySQL Web Development*. Indianapolis. SAMS.

Zandstra, M. (2005) *Teach Yourself PHP in 24 Hours, 2nd Edition*. Indianapolis. SAMS.

Websites

Codewalkers

<http://codewalkers.com/>

Offers a wide range of resources on PHP and MySQL including tutorials

PHP

<http://uk.php.net/manual/en/introduction.php>

An introduction to PHP from the official website which includes a very useful introductory tutorial.

MySQL Tutorials

<http://www.php-mysql-tutorial.com/>

A series of tutorials on how to use PHP and MySQL to create and administer databases.

PHP/MySQL Tutorial

<http://www.webmonkey.com//programming/php/tutorials/tutorial4.html>

A relatively straightforward introduction to PHP and MySQL, including data validation.

ASP.NET / ASP

The resources below refer to ASP.NET which has been designed to supercede ASP. However, at the time of writing ASP remains a commonly-used server-side technology and a wide range of resources are available offering information and tutorials in its use.

Books

Duthie, G. A. and MacDonald, M. (2003) *ASP.NET in a Nutshell*. Sebastapol, CA. O'Reilly.

Hart, C., Kauffman, J., Sussman, D. and Ullman, C. (2005) *Beginning ASP.NET 2.0*. Indianapolis. WROX.

Kittel, M. A. and LeBlond, G. T. (2004) *ASP.NET Cookbook*. Sebastapol, CA. O'Reilly.

Martinez, J. and Parnell, R. (2003) *ASP.NET Development with Dreamweaver MX*. Berkeley, CA. Peachpit Press.

Mitchell, S. (2003) *Teach yourself ASP.NET*. Indianapolis. SAMS.

Walther, S. (2003) *ASP.NET. Unleashed*. Indianapolis. SAMS.

Websites

ASP.NET Quickstart Tutorial

<http://www.asp.net/QuickStart/aspnet/Default.aspx>

Detailed tutorials on using ASP.NET including information on how ASP.NET controls are used with code examples.

4 Guys from Rolla

<http://www.4guysfromrolla.com/>

Searchable resource with articles and tutorials on specific aspects of ASP.NET.

CGI/PERL

Books

Colburn, R. (2003) *Teach yourself CGI*. Indianapolis. SAMS.

Guelich, S., Gundavaram, S. and Birznieks, G. (2000) *CGI Programming with Perl*. Sebastapol, CA. O'Reilly.

Websites

CGI Programming 101

<http://www.cgi101.com/>

Tutorials aimed at beginners with information on how to set up a development environment using CGI/PERL and how to process forms and write data to files.

CGI Made Really Easy - or, Writing CGI scripts to process Web forms

<http://www.jmarshall.com/easy/cgi/>

Basic introduction to collecting and formatting information from forms.

Elated

<http://www.elated.com/tutorials/programming/perl/cgi/>

Tutorials covering a basic introduction to CGI programming with PERL along with issues such as validation and emailing.

ColdFusion

Books

Brooks-Bilson, R. (2003) *Programming ColdFusion MX*. Sebastapol, CA. O'Reilly.

Camden, R., Chalnick, L., Buraglia, A. C. and Forta, B. (2005) *Macromedia ColdFusion MX 7 Web Application Construction Kit*. Berkeley, CA. Macromedia Press.

DeHaan, J. (2004) *ColdFusion Web Development with Macromedia Dreamweaver MX 2004*. Berkeley, CA. Apress.

Mohnike, C. (2003) *Teach Yourself Macromedia ColdFusion in 21 Days*. Indianapolis. SAMS.

Websites

Macromedia's Support Centre for ColdFusion - Tutorials

http://www.macromedia.com/support/coldfusion/tutorial_index.html

A wide range of tutorials. Part of the ColdFusion Support Center which includes resources, technical notes and a forum.

EasyCFM Tutorials

<http://www.easycfm.com/tutorials/index.cfm?dirView=True>

A comprehensive range of ColdFusion tutorials.

JSP

Books

Bergsten, H. (2003) *JavaServer Pages, Third Edition*. Sebastapol, CA. O'Reilly.

Brunner, R. (2003) *JSP: A Practical Guide for Programmers*. San Fransisco, CA. Morgan Kaufmann Publishers.

Holzner, S. (2002) *Teach Yourself JavaServer Pages in 21 Days*. Indianapolis. SAMS.

Perry, B. W. (2004) *Java Servlet & JSP Cookbook*. Sebastapol, CA. O'Reilly.

Websites

Caucho JSP Tutorials

<http://www.caucho.com/resin-3.0/jsp/tutorial/index.xtp>

Covers topics including form processing and emailing form contents.

JSP Tutorial

<http://www.jsptut.com/>

Series of tutorials covering the basics of JSP and dealing with forms processing, databases and emailing. States that users should have a knowledge of HTML and Java.

JSP Olympus

<http://www.jspolympus.com/JSP/JSP.jsp>

Comprehensive range of tutorials.
