WebExp2
Experimenter's Manual

Neil Mayo
nmayo@inf.ed.ac.uk
School of Informatics
University of Edinburgh

Martin Corley
martin.corley@ed.ac.uk
Department of Psychology
University of Edinburgh

Frank Keller
keller@inf.ed.ac.uk
Human Communication Research Centre
University of Edinburgh

www.webexp.info

24th October 2005

Contents

List of Figures

1 Introduction

WebExp2 is a toolbox for conducting psychological experiments over the World Wide Web. This section provides an overview of WebExp2 and describes the main features available, along with the advantages of conducting experiments over the web.

Note: Please refer to the glossary in section 10 for explanation of terminology used in this document. Some terms, though familiar, will have specific imnport in the context of WebExp2.

Client-Server WebExp2 is a client-server system, consisting of two separate programs which interact with one another.

- the **Server** is an `application` which you run on the web server machine which hosts the experiment files. It waits for client applets to connect to it and request experiments, whereupon it provides (serves up) the experimental files, and manages the results that are returned.
- the **Client** is an `applet` which runs in the browser of the participant. It connects to the server to get the experiment, administering it to the participant and returning results to the server.

Java & XML WebExp2 is written in Java, and uses XML as the description language for a) defining experiments and b) storing results. There are two parts to the system: the server (an application), and the client (an applet). Java is specifically designed to be platform-independent and is particularly suited to running across the internet. So the WebExp2 server can run on Windows, Linux, Unix, MacOS, and anyone with a browser can run the experimental client. As a data-description language, XML is standardised, flexible, and supports simple validation of the data.

In order to get WebExp2 ready for use, you need to accomplish a handful of tasks. It is a good idea to get the system working with one of the supplied paradigms, and then you can go on to designing your own experiments to run on the system.

1.1 Features

Experiments are designed around a sequential stimulus/response model. This model can be seen as a 'questionnaire' where you provide a series of 'questions', each of which involves the presentation of stimuli and the collection of responses (via the presentation of a set of response options).

1.1.1 Implemented

These features are implemented in the current release of WebExp2 and form part of the core functionality of the system.

Paradigms There are a handful of sample paradigms implemented and included with WebExp2. These can be easily modified to create your own experiments, or can be studied to provide tuition in how to achieve particular effects with the system. The following paradigms are available:

- the Stroop task (parametric 2-condition)
- attractiveness and pupil size (nonparametric 2-condition)
- discriminant reaction times

There are also a couple of demonstrations, one showcasing the available features, and the other demonstrating the layout possibilities.

User-defined experimental designs In addition to these supplied experiments, it is straightforward to design new experiments and implement other paradigms, without having to know anything about the code behind the system. You don't have to be a programmer to design experiments for your own purposes. All you need is a plan, and an understanding of the XML language used to describe experiments. This language is introduced in detail in this manual, and is reasonably intuitive for non-technical users. You will find it particularly straightforward if you have ever used HTML to design webpages.

Of course if you want to get stuck into the source code and extend the system, that is possible too — please consult the Developer's Manual for guidance.

Templates Instead of explicitly describing every step in the sequence of a repetitive experiment, it is possible to describe one or more steps that will be repeated, importing different materials into each according to your requirements. So for example to implement a Stroop experiment you only have to define the layout once, and can separately define the words and colours that you want to use in each step.

Separation of form and content To complement the abstraction of the form of your experiment into templates, the materials or stimuli you wish to use (the content) are specified in separate documents. Advantages to this approach are that it simplifies experiment descriptions, while making it clear what your stimuli are and how they relate to one another. Stimuli can also be grouped into blocks, and rearranged without affecting the structure of the experiment. This should make it easier to achieve the result you want from your experimental design.

Timing WebExp2 has a timing unit which can be used to measure time intervals so that a) response times can be collected to monitor the onset and duration of responses, and b) timing may be manipulated to delay or limit the appearance of stimuli.

All timings are recorded in the results along with a confidence interval which can be consulted to determine whether the timing interval was administered accurately. This confidence interval is calculated using an algorithm described in Eichstaedt (2001), and should allow an experimenter to filter out any unreliable data which is due to inaccuracy of timing measurements over the web using Java.

[describe internet timing limitations]

Configurable visual properties WebExp2 gives you control over the visual properties of your experiment. Font properties and background and foreground colours can be specified for each stimulus, or default properties can be specified which may be overridden.

Layout options WebExp2 provides a flexible visual layout manager which gives control over both horizontal and vertical alignment and positioning. You can define proportional row heights and column widths and then position your visual stimuli with respect to this grid.

Stimuli There is a variety of stimulus types that may be presented in your experiment, including images, text, animations, and blocks of colour.

Responses There is also a variety of response types. You can collect responses using text inputs (with input constraints), selection lists, buttons, and key or mouse input.

Note on Stimuli and Responses Stimuli and Responses will be referred to generically as Components, as they are both components of your presentation. They are called components in the xml description language too. Further stimulus and response types may be added quite straightforwardly if you know a little Java — see the Developer's Manual for more details.

Randomisation It is possible to randomise sections of your experiment, and to reorder them by hand. Any reorderings are recorded in the results. WebExp2 currently supports only free randomisation and blocked randomisation – we hope to add constrained randomisation in the near future. Guidance concerning creating new randomisation types can be found in the Developer's Manual.

Persistent values You can use responses you have already collected in later sections of the experiment, allowing you for example to display reference responses in magnitude estimation experiments. *This feature will also provide for branching of the experimental sequence with a little further development.*

Logging Comprehensive logging for both client and server, recording potential and actual problems.

Server features The server is easy to use, providing automated administration and enforcing the separation of each experimenter's respective experiments and results.

- Directory-structured data management and storage
- Security for subject information and data
- Multiple experimenters can share the same server

Other features There are various features to enhance the usefulness of experiments and to provide feedback.

- Simple questionnaire navigation — subjects can tab between fields, and press return in a field to advance
- Mandatory responses — ensure you get input before the participant can continue
- Progress indications — display the name of the experiment, and progress counts
- Informational methods for use by javascript

1.1.2 Desirable

This section describes features which would enhance the usefulness of WebExp2 and which we would like to add in future stages of development. WebExp2 has been designed with some of these in mind and should be extensible without too much complication. Further description of these potential features can be found in the Developer's Manual.

Server features The server will handle the randomisation and counter-balancing of experiments based on the specification, thus automating the creation of multiple versions of an experiment. Improved data management and storage. Perhaps also a GUI.

Validation There are various points where automatic validation would be useful and is not yet implemented:

- Participant authentication
- Experimenter login authentication — for access to experiments and results
- Validation of results — for example to check for junk results
- Automatic handling of invalid or incomplete response data
- Validation of an experimental design

Further randomisation Constrained randomisation and automatic counter-balancing based on the experimental description.

Client (experiment) features Provide an indication of progress while the client is initialising and gathering resources.

More stimulus types We will write extensions to allow acoustic stimuli, add new stimulus types such as audio. Eventually video may be added, and potentially other media.

More response types We will develop some more complex response options such as Likert scales, grid responses, selections, and arbitrary scales. This will remove the need for them to be designed ad-hoc for each experiment.

Greater control over the participant's environment Control over the visual environment and experiment interruptions, allowing one to minimise problems encountered from subject dropout or junk results:

- Full screen display to eliminate background distraction
- Monitor for dropout or loss of focus such as window-switching

Conditional branching Extending the ability of subject input to affect ensuing sections of the experiment, we will allow the output of a slide to decide the identity of the subsequent slide, allowing varied paths through an experiment.

Improved timing Finalise fully-featured and precise timing, allowing close control over the timing of stimulus events.

Arbitrary action invocation In addition to allowing the timed display/withdrawal of a stimulus, it will be possible to make a stimulus perform any other arbitrary action on a timed signal (as a simple example, to change background colour or text). This will make possible the creation of more dynamic types of stimulus, for example counters.

These last three features will allow WebExp2 to provide better support for self-paced reading studies along with a wide variety of other experimental paradigms.

Graphical frontend for experiment design The creation of a user-friendly interface and tools for experimental design and evaluation will further simplify the design task, reducing the experimenter's contact with xml.

Analysis tools Further tools will be provided for statistical analysis and data processing of experimental results, including common statistical normalisations.

Internationalisation and accessibility A major enhancement to maximise the usability and universality of WebExp2 will be to add multi-language support, by providing localised versions of the system, and adding support for international languages and fonts.

1.1.3 Known problems

- Cannot resize pictures ..
- Pictures (anim gifs in partic..) sometimes aren't shown (at least viewing on Linux) - more research needed!
- IE does not seem to provide browser to javascript

1.2 Web-based experimentation

This is a short discussion of the pros and cons of Web-based experiments. For more a more extensive treatment of Internet experimentation, see Hewson et al. (1996) and Reips (2002).

1.2.1 Pros of Web-Based Experiments

- The ubiquity of internet access provides a large pool of potential subjects for the experiment.
- A variety and multiplicity of results can be collected.
- Saves time for the experimenter and the subject.
- (Putative) higher response rate from subjects and no danger of subjects agreeing to do it and then not showing up.
- Questions are presented in order, and the subject has no chance to change their answer to a previous question after seeing a later one.
- Subjects cannot accidentally fail to respond to an experimental item (the software will not allow the subject to just press the return key).
- Response timing may prove useful to check that the experiment has been completed, or even for evaluation.
- Subject authentication is better than if one allows subjects to take a questionnaire home.

1.2.2 Cons of Web-Based Experiments

- Subjects are self-selecting. Conducting the experiment over the Web restricts the subjects to those who are competent at using Web browsers, which may be an unrepresentative sample.
- Subject authentication is not as good as if the experiment is done under laboratory conditions.

2 Setting up an environment for the system

2.1 Java

JRE for the server To run the WebExp2 server, you will need a Java Runtime Environment (JRE). If you wish to develop the program further by writing your own components or modifing existing code, you will need the Java Development Kit (JDK). This includes a JRE too, but is a larger download.

Java Plug-in for the client Your participants will only need a Java Virtual Machine (JVM), the most basic unit of Java — this will be part of any Java implementation they install, but is most easily obtained as the Java Plug-in, from `http://java.com`. Most browsers will simplify the process of locating and installing the plugin if a participant has no Java installation when they visit your experiment.

Java versions Both yourself and your participants will need Java of at least version 1.4. The program uses features which are unavailable before Java 1.4 and will not work without it. Java 1.4 has been available since the end of 2002 and the latest version (as of the time of writing, June 2005) is 1.5, with which WebExp2 is also compatible.

2.2 Downloading Java

You can get `Java J2SE 1.4` from
`http://java.sun.com/j2se/1.4.2/download.html`.
Download either the JRE, or the SDK if you want to develop WebExp2.

Alternatively you can get `Java J2SE 1.5` (also known as `Java J2SE 5.0`) from
`http://java.sun.com/j2se/1.5.0/download.jsp`.
Download either the JRE, or the JDK if you want to develop WebExp2.

You can also download the documentation from these links.

`Java Plug-in` can be downloaded from
`http://java.com`
and will let your participants run your experiment. The standard download is the latest version, `Java 5`.

2.3 Operating system requirements

Being written in Java, WebExp2 is theoretically platform-independent. As far as we know there are no problems running the server and the experiment client on Windows, Linux or Unix-flavoured systems, and MacOS. The filesystem must be capable of filenames longer than 8+3 in order to manage the results data.

2.4 Unzipping, setting permissions

WebExp2 is provided as a zipped tar archive (with the extension `tar.gz`). The following steps have to be carried out to install it on your local host:

Unzip the file; it will give you a folder `release`, containing a manual, various sample experiments and their resources, and the WebExp2 jar file which contains all the Java class files. There is also a `README` file describing the contents in more detail.

Eventually you will place some of the contents of this directory on your web server, but you can test your experiments locally without setting up a web server. **Note:** you can rename this directory to whatever name you find convenient, but do not rename any directories or files within this directory.

You can unzip with either command line programs (Linux) or programs with a visual interface such as WinZip (Windows), as long as the zipped folder structure is maintained.

3 Running sample experiments

This section describes how to get the WebExp2 server and client running so you can try out some of the sample experiments.

3.1 Starting and maintaining the Server

3.1.1 Configuration files

Upon startup, the server reads in configuration files which tell it who will be using it (the experimenters) and what experiments these experimenters are running. Thus it initialises itself with information about the currently available experiments — the server is capable of supplying any of these experiments to clients which request them.

When you first unpack WebExp2, the server is set up with default experiments which we have supplied; you will need to edit the configuration files later to tell it about yourself and your experiments (see section 6.1).

3.1.2 Starting the server

When you have unpacked WebExp2, you can start the server by issuing the following commands at your shell/console/DOS command line.

Figure 1: Server startup command and options.

The server is started with the command:
`java -cp webexp2.jar server.Server [OPTION]...`

Options are prefixed by a hyphen - (for example `-p9876`) and include:

`p`	port	which port the server should use for communication

3.1.3 Maintaining the server

The server outputs log messages (see section 8.3) to the command line, so you can keep an eye on its current state. Of course, while the server runs, results and log files for completed experiments will accumulate in the specified directories. Therefore you should not leave the system running indefinitely but be aware of your accumulating results and the available space on your hard drive.

Server log messages are currently not saved, but just echoed to the screen — serious errors that crash the server will leave a record on the console. Running information however will of course fall prey to the limited screen buffer of a console. In the future logging will be more sophisticated and will involve logging server messages to managed files.

You can stop the server application at any time by pressing `ctrl-c`, which interrupts Java. However, you can also interact with the server application while it is running, and we have supplied commands to stop the server more gradually.

3.1.4 Server commands

The server accepts commands while running in the console, allowing the display of summary statistics. Available commands are listed in table 2 To invoke a server command, type a letter at the console followed by `return`. Try `h` first to see the available commands.

Figure 2: Server runtime commands. These are initiated by typing the letter followed by return.

`q`	(Q)uit	hard quit — close the server, terminating connections
`w`	(W)ait	quit the server, after waiting for connections to close
`d`	(D)ump	dump current state records to file (also done automatically before quitting)
`i`	(I)nfo	print summary server state information
`c`	(C)onnections	show current connection details
`e`	(E)xperiments	show all experiments
`a`	(A)ctive	show active experiments (those available to clients)
`r`	(R)unning	print details of running experiments (those with connections)
`h`	(H)elp	print command list

This feature was added primarily to allow a clean shut down of the server while it is administering experiments. The `quit` and `wait` commands save current records before stopping the server, which is essential for maintaining experiment completion records, and these commands should be used in preference to `ctrl-c` for stopping the server.

The `wait` command additionally attempts to wait for running experiments to complete before stopping the server, *though this is not yet implemented.*

You can also use `ctrl-c` to perform an unconditional quit. This stops Java instantly and is useful if you have a problem. However it does not save records.

3.2 Storage of results and logs

As clients connect to your server and administer the experiments, information is returned each time a subject completes an experiment. You should get incrementally numbered files created on your system.

Relative to the `release` directory, you will have a data directory `data/experimenter/experiment` for your experiment, and the following directories will be created within this for storing results: `results`, `subjects` and `logs`. Each will contain numbered files in the format `experiment+number.xml`.

Each time someone participates in an experiment, a unique ID is generated to identify that instance of the experiment. The IDs are saved to the `uidmappings` file in the `data` directory, along with information identifying the experiment and the number used to save the results files. Results files are saved in XML format and are easily processed to extract the information you want.

3.2.1 Results

The results file records all response information for each 'question' in the experiment. This ranges from a simple button press to textual input and timing information. Identifying properties are also output so you know where each response came from, even with a randomised experiment.

3.2.2 Subjects

Subject information is separated from the experimental results so that anonymity may be enforced. Any information collected in a 'subject' stage (see section 5.5.3 for stage types) will be stored in the subject file along with an ip address and information about the computer environment of the participant.

3.2.3 Logs

All non-fatal errors, assumptions, erroneous experimental descriptions and defaulted properties are logged, and the log is sent to the server at the end of a successfully completed experiment. The log can therefore be consulted to ensure for example that an experiment executed as expected. Similarly, the log may give insight into unexpected results. Log files are saved in text format with a message on each line.

3.3 Running the client applet

When the server is set up to supply a particular experiment, you can view it by opening an HTML page which has an `<applet>` tag naming the experiment. You can either open the page in a browser which has the Java plug-in installed, or you can view it in the Java `appletviewer` program, with a command like `appletviewer stroop.html`.

3.3.1 Demonstrations and example paradigms

Some sample implementations of well-known paradigms are provided. These should a) provide a good source of examples of the usage of the experiment language, and b) provide a base which you can adapt to your own needs. The demonstration experiments attempt to introduce a majority of available features and show how their use can be varied to get different results.

3.3.2 How to use a paradigm

The server is already configured to use the supplied experiments, so just start the server (see section 3.1) and then view the experiment you want by opening its associated HTML page from the `release` directory.

Have a look at the experiment's directory `data/default/experiment`, in which you will find the XML file describing the experiment. Results, logs and subject records will also accumulate here as experiments are completed. You can consult these simple examples and their XML descriptions while reading the following sections, which introduce the elements of an experiment and the language we use to describe them.

3.3.3 How to adapt a paradigm

At first, you can experiment with changing values and adding new components in the default description files. You can thus get used to the language and try some of the possibilities without having to worry just yet about the other aspects of creating an experiment.

When you create your own new experiments, you will create a separate description for your experiment in XML, and will adapt the server configuration files so it knows who you are and what to do with your experiment. While learning how to use the system however, it may be easier to play around with the XML of the existing experiments to understand how it works.

The following section 4 will describe the elements that are available to you for creating your experiments.

4 Implementing an Experiment

WebExp2 provides for the specification of experiments and thus is not restricted to particular paradigms. You can design experiments based on a sequential, questionnaire (Stimulus/Response) framework, with control over timing, subject responses, and faulty data.

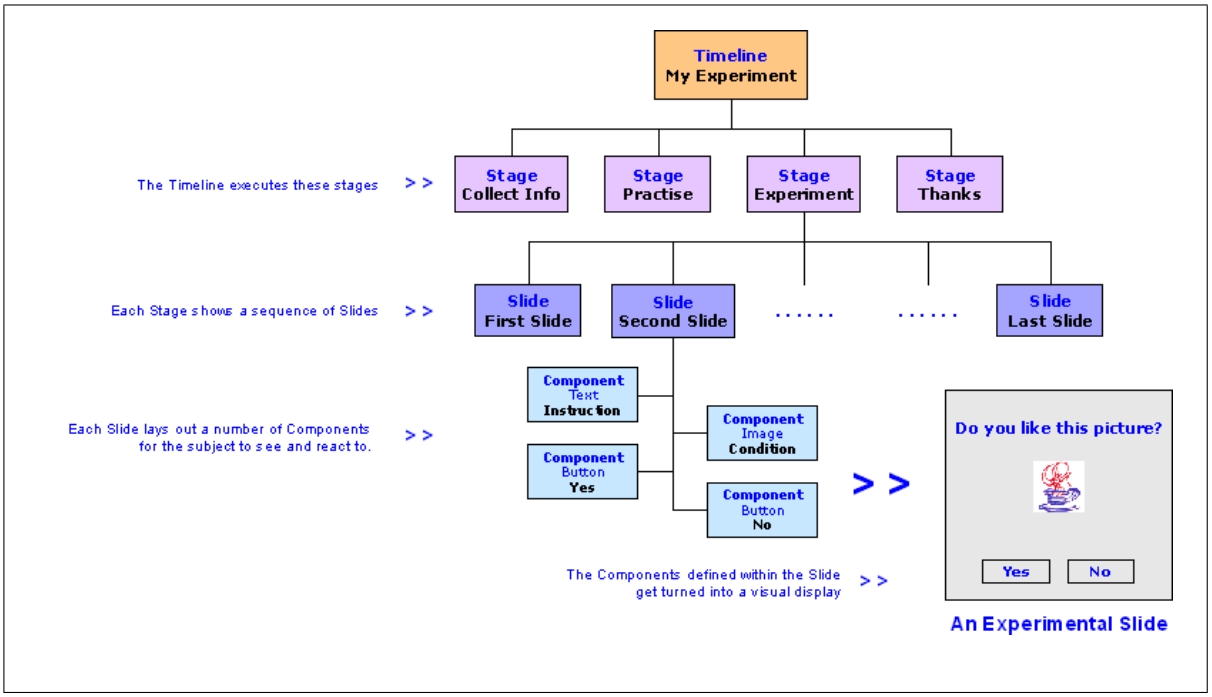
This section introduces the essential concepts in a WebExp2 experiment, summarises XML usage and describes the components that are available for creating experiments.

Note: As you read through this manual, you will come across certain pieces of terminology which may be familiar but have a specific import with respect to WebExp2. You can get a precise definition of these terms in the glossary (section 10).

4.1 Structure of an experiment

Timeline To implement an experiment, you have to describe a `Timeline`, which is a sequence of `Stage`s along with some properties. Each `Stage` is a stage of the experiment, such as introductory stage or practise stage, and consists of a sequence of individual `Slide`s. Each `Slide` consists of some properties of its own in addition to a set of `Component`s, which also contain properties.

Figure 3: The Timeline hierarchy used in an experiment description.



Description language You describe these components and their properties with a description language we have defined. This language is interpreted by the client and the information it yields is used to create the sequential display for your subjects.

4.2 XML

The description language is defined in XML (eXtensible Markup Language). This format will be familiar to anyone who has written HTML pages, as HTML is a subset of XML. However XML is stricter, as will be explained in the following section.

Each experimental component `myComponent` is defined by everything that appears between a pair of tags `<myComponent>` (the opening tag) and `</myComponent>` (the closing tag). All objects and properties are defined in this way, with angle-bracketed 'tags', and these are called `elements` of the XML document.

The content of an element can be text or further elements, and in this way an entire tree structure can be defined, each element describing either a single property value or a complex object consisting of a collection of sub-elements. For example our font property is complex, consisting of the three properties (sub-elements) face, size and style.

4.2.1 Writing XML

Every XML document starts with the XML declaration, which will look like the following:

```
<?xml version="1.0" encoding="UTF-8"?>
```

This is followed by a number of XML elements. There must be a root element which contains all the other elements, so an XML document may look somewhat like this:

```
<?xml version="1.0" encoding="UTF-8"?>


<!-- This is a comment -->
<the_root_element>
   <element1>Hello</element1>
   <element2>
      <element3>World</element3>
   </element2>
</the_root_element>
```

Each pair of tags `<tag></tag>` define an `element`, and the information between the tags is the element's `content`. If this is just text without sub-elements, it may be called the tag's `value`. A tag may also have `attributes`, which are properties of the element which are defined within the opening tag. An example from HTML is the `src` attribute of the `<img>` tag, which describes the source of an image: `<img src=''mypic.jpg''></img>`.

Note: The information in an attribute might just as well be placed in a similarly-named child element of the element concerned. There are no real rules about whether textual data related to an element should be included in attributes or child elements, and we have tried to use child elements in WebExp2 to avoid confusion. However there are some attributes in the specification; we have introduced these where the data concerned feels more like a property which is internal to the element, or where it makes the xml look cleaner.

XML elements must observe some simple rules to keep the document 'well-formed'. These are:

- all elements must have a closing tag (if you have an empty element, you can write `<element/>` instead of both opening and closing tags)
- tag names are case-sensitive and cannot contain spaces
- white space is preserved
- elements must be properly nested (you must close an element before close its parent element)

- comments can be written like this: `<!-- This is a comment -->`
- attribute values must always be quoted

A fuller description of XML and the precise rules that govern its use can be found in the excellent tutorial at http://www.w3schools.com/xml

4.2.2 Reading XML

As every property or object must be defined within these matching opening and closing tags, XML results in a rather verbose experimental description. However it is straightforwardly legible with a little practise, and particularly with indentation and a code-colouring text editor which will show you the distinction between your property tags and their values.

An excellent and versatile editor which will colour XML for you is `Scite`, available for free from http://www.scintilla.org.

Using XML also means that it is simple to validate and transform both experimental designs and results, and makes it relatively legible to a non-technical user.

4.3 How to write an experiment in XML

To implement an experiment you must write an XML file describing the sequence of the experiment, and optionally one or more resource files which describe your stimuli. An experiment is described by a `Timeline` which consists of one or more `Stage`s, each of which contains one or more `Slide`s. The `Timeline` therefore is the root element of the XML document. The next section describes the function and behaviour of the elements (stages, slides, components) which you define within the timeline.

Observing the XML 'well-formedness' rules (section 4.2.1), you define your stages and slides along with their properties; filling the slides with components, which act as stimuli and collect responses for you. Then you can run the experiment, and your sequence of slides gets displayed.

If you describe an unknown or inapplicable property within your XML, it will be ignored during the experiment. However the document must be a 'well-formed' XML document. Some information is output to the logs about illegal properties and values, so you can clear up anything that may be providing unexpected behaviour.

Note: In 2005/6 we should be developing a front-end for WebExp2 which will simplify the process of designing slides, shielding the user from the repetitive task of defining the XML by hand.

4.4 Description of the Slide

The slide is the area where you display the components (stimuli and responses) that you want to present in each step of your experiment. Think of it as a point in the sequence of your experiment, and as the page where you will *present* the stimulus to the subject, whether that is a picture, some text, an audio clip, or an animated image; and where you will also present components for collecting responses and input.

Components are arranged on the screen by defining a layout for your slide, specifying row and column sizes and positioning components within the resulting grid. You can specify the visual properties of `font` and `color` when describing a slide, which will provide defaults for any visual components you place onto the slide.

Each slide is used to present stimuli to the subject. These stimuli can be defined within the slide description, but it is more sensible to separate the changeable content of your experiment from the form. You can do this by writing separate files which describe your stimulus resources, and importing these into your slides. This is described in a later section.

4.5 Description of available Components

This section describes the types of stimulus and response components which are available to you in v 1.0 alpha. It is important to understand what these are and what properties can be defined for them before you design an experiment.

The components are listed by name; this name is used in the XML to signify which type of component you want. Currently there are only visual components — all visual components take properties including width, height, and background and foreground colours. Additionally you can specify font properties for textual components.

4.5.1 Text

The `text` stimulus allows you to present some text on the slide. You can provide no text and thus use this stimulus as a coloured box. You can specify line returns within your text by typing $\backslash n$.

4.5.2 Image

The `image` stimulus allows you to display a jpeg, png or gif image on the slide. You can stretch the image by specifying a size. Animated gifs should work, but cannot be resized.

4.5.3 Button

The `button` stimulus provides a clickable button. This can be used simply to advance the slide, but also to provide a set of options. All buttons when clicked will advance the slide, but the button name is recorded and so this stimulus can be used as a response.

4.5.4 Input

The `input` component is a response component, and allows you to get textual input from the subject. You can specify which characters are allowed and the maximum length of the input.

4.6 Responses and reusing values

You can use the responses you collect to affect later slides. Any value which is saved to the results can be retrieved and used. Currently the value can only be used as a property value in a new component, but in future could be used to decide conditional paths through an experiment.

Note: This feature is not fully functional if you use templates to describe your experiment. See section 5.5.4.

4.7 Randomisation of an experiment

It is possible to specify parameters which affect the ordering of slides within a stage. Note that this is only possible *within* a stage; the stages themselves remain in the order in which they are described in the XML, and there is no way to randomise across stages. There are two ways to reorder the sequence of an experiment.

4.7.1 Specifying an order

The first is by specifying an `<order>` tag within a stage. This is useful if you want to change the ordering in a deterministic way, i.e. if you know what order you want the slides to be in. The `<order>` tag makes it easy for you to do this without reordering slides by hand. You can change the order for testing for example, and comment out the tag if you want to revert to the natural ordering.

4.7.2 Free randomisation of slides

The second approach is to add a `<randomise/>` tag to a stage, which tells the client to randomise whatever sequence of slides it finds in that stage.

Note: Any reordering is output to the results for future reference.

4.8 Randomisation of resources

Another method of randomisation is available if you are importing stimulus resources into your experiment. You can define the structure of your experiment (sequence of slides, repetitions and so on) and then separately randomise the resources which you put into that structure.

Currently only free randomisation and blocked randomisation are supported. In future we plan to add constrained randomisation, along with automatic counterbalancing for conditions.

To randomise a set of resources, you simply add a `<randomise>` attribute to the resource set when you import it. This is described in more detail in section 5.4.4 on importing resources.

Note: Any reordering is output to the results for future reference.

5 XML description language reference

This section introduces the language used to describe the essentials of an experiment, including precise information on the options available for properties. The approach is to take you gradually through the whole process of designing an experiment structurally from the top down, such that you may like to refer to this section when designing an experiment for the first time. We also suggest you refer to the XML of one of the sample experiments while you read this section.

5.1 Separation of form and content

WebExp2 allows you to separate the stimulus resources which you use within your experiment from the structure of the experiment itself, by defining them in external files. So in addition to the file we are creating here, you will create one or more resource files which you can then import into the main experiment description.

Advantages to the separation of form and content include the following:

- Repetitive slide sequences can be specified more simply with a single slide (called a `template`) along with a repetitions parameter, leading to vastly simplified experiment description files
- It is much clearer what your stimuli are and how they relate to one another
- It is simple to rearrange or replace your stimuli without inadvertently affecting the structure of your experiment
- Stimulus resources can be grouped into blocks or classes which are more easily randomised
- Stimulus resources which are imported can override default properties of the slide components, adding to the expressibility of the stimuli

> Next: **Create an experiment description file**
> First you will need to create an experiment description file, containing a timeline. This describes the sequence of the experiment and the various items that you will show to your subjects.

5.2 Timeline

The `Timeline` is the root element of the experiment description. A timeline is composed of a sequence of stage elements, and other elements which describe properties of the timeline. The properties applicable to a timeline are specified in table 4. You should define a title, and also resources if you are importing them, and an imagebase if you are using images. Then you have to define stages, slides and components.

Figure 4: Timeline property tags.

Tag name	Content
`title`	a title for the experiment; will be shown at the top of the applet during the experiment and is used in results
`imagebase`	a path relative to the applet (initially, the `release` directory) where images are stored
`images`	a list of individual images which are to be loaded for later use in the experiment
`resources`	a list of resource sets which are to be loaded for later use in the experiment

So the start of your experiment file might look like this:

```
<timeline>
   <title>My New Experiment</title>
   ...
```

The `<resources>` and `<images>` tags are described in more detail in the following section. Note that images can be imported in two ways:

- With an `<images>` tag – this allows you to name images so you can say precisely which image you want to place into your experiment design, and to use it multiple times. (This is useful for example to show a logo or repeated reference image.)
- With a `<resources>` tag – in this case your list of images is stored in an external resource file, and when you want to use an image in your experimental design you refer to the resource set rather than a specific image. (This is useful when you want to import images as stimuli, and is necessary to randomise the order of your images.)

> Next: **Import named image resources**
> To use particular named images in specific places in your experiment, you will list them in an `<images>` preamble at the start of the timeline.

5.3 Image files

If you have a small number of images which you wish to use in specific components in your design, it will be easiest to use the `<images>` tag to list them. **Note:** You can still use resource sets in addition to these images, for importing randomisable *lists* of images.

To use image resources, you should define an `imagebase` element which sets the directory where the images may be found, and then a list of images you want to use from that directory. Figure 5 shows an example of importing individual images.

The `images` element contains a number of elements of the form `<name>filename</name>`, where `name` is a name by which you can refer to the image to include it in your slides, and `filename` is the full name of the file containing the image. This file can be a JPEG, PNG or GIF image.

You can later import any of these images repeatedly by defining an image component with a content property such as `<content>earth</content>`.

> Next: **Create external resources**
> If you are importing your stimulus resources, you will need to a) write some resource files which list your resources, and b) specify these files in your timeline.

5.4 Resource files

Resource files describe the changeable parts of your experiment – the content. This comprises such things as images or sentences which you want to present to the subject. It will also include audio files when audio is implemented. You *can* include text resources directly in the components of your timeline, using the `<content>` property of components. However if your experiment uses a repetitive sequence of similar slides, it would be more sensible to use resource files, for reasons elaborated in section 5.1.

Figure 5: Defining individually-named images to use in an experiment.

```
<?xml version="1.0" encoding="UTF-8"?>

<timeline>
   <title>My Experiment</title>
   <!-- Resources for preloading -->
   <imagebase>images/</imagebase>
   <images>
      <earth>earth.gif</earth>
      <calvin>calvin.jpg</calvin>
      ...
   </images>
   <resources>
   ...
   </resources>
   ...
</timeline>
```

The basic scheme is that you list your resources in one or more files, then tell the timeline to use resources from the files, and how to randomise them. Then you can import a resource wherever you want to display one in your timeline sequence.

Resources in the files can be grouped into blocks. Randomisation can then be performed within or between these blocks, as well as freely over the entire set.

5.4.1 Resource file format

A set of resources is defined in an XML file which contains a number of blocks, each of which contains a number of resources. A resource takes the form of a piece of text, which may be a sentence or a file name – this text ends up in the `<content>` tag of a component, and is interpreted based on the type of the component. See table 16 in the components section for further information on components. A sample resource file is shown in figure 6.

Each block and each resource should have a unique `id` attribute; this is to save conflicts and also to give you something in the results which can be used to identify which resources were used in randomised experiments.

5.4.2 How to use blocks

Blocks are used to group related resources – your resources may be related by their purpose in your experiment, their subject, their class or type, or innumerable other possible categorisations. It will sometimes also be necessary or desirable to segregate your resources using separate resource files.

As an example, if you have images of faces and of feet, you might group them by type (a faces block and a feet block) or by person (a block per face-feet pair). You will know what groupings make sense to your experiment. The important thing to consider is how the groups will be randomised and how they will be imported into your experimental template. Essentially, each resource set is treated as a single list of resources which is randomised according to the blocks and parameters you specify. Every time your timeline sequence requests a resource from a particular set, the next resource in that set's list is imported.

Figure 6: A resource file defining blocks of images.

```
<?xml version="1.0" encoding="UTF-8"?>

<resources>

<block id="images A">
   <resource id="01.01">image1.jpg</resource>
   <resource id="01.02">image2.gif</resource>
   ...
</block>

...
</resources>
```

So for example a block could be used to group an image filename with an image title – a number of file/title pair blocks can be created, then the order of the blocks can be randomised (between blocks) and each slide imports two resources from the resource set; one into an image component, and the second into a text component which describes the image displayed.

5.4.3 Overriding component properties

As well as defining the `id` attribute of each resource, you can define an arbitrary number of additional attributes. These additional attributes represent properties of the component into which the resource gets imported, and can be used to override any default properties of that component. An example of this approach is shown in figure 7.

Figure 7: Resources which define additional attributes to override defaults in the timeline sequence.

```
...
<block id="word-colour">
   <resource id="01.01" fgcolor="yellow">BLUE</resource>
   <resource id="01.02" fgcolor="red">GREEN</resource>
</block>
```

This illustrates one way in which a Stroop task could be constructed. If the resources in this example are imported into text components, two different results would be produced; one with the word `BLUE` in yellow, the other with the word `GREEN` in red. An experimental template only needs to describe a single text component with an `import` tag, and on each slide with this component, the result will be different.

Any other properties of the target component may also be overridden, such as `height`, `width`, `bgcolor`, even `type`. All applicable properties you define in a resource will be coerced into the target component; any which are not appropriate will have no effect. If you do not specify content for a resource but instead leave it empty, for example `<resource id="01.02" fgcolor="red"/>`, the experiment will use whatever is already defined in a `<content>` tag in the experiment file. This behaviour can be used to display the same word repeatedly but in different colours. The word is defined once in the text component in the timeline, while the changing colours

are defined in the resources file.

5.4.4 Importing resources to your timeline

Resource sets are loaded by the timeline, so you must declare your resource files in the timeline properties. When you have defined your external resource files, you tell the timeline which of these files are to be used during the experiment. Resources specified here will be preloaded and cached before the experiment is started, so there are no delays in using them. This is also the place to specify an ordering or randomisation.

Figure 8: Defining the resources to use in an experiment.

```
<?xml version="1.0" encoding="UTF-8"?>

<timeline>
   <title>My Experiment</title>
   <!-- Resources for preloading -->
   <imagebase>resources/art</imagebase>
   <resources dir="myresourcefiles">
      <sentences randomise="between">sentences.xml</sentences>
      <artists randomise="free">art.xml</artists>
      ...
   </resources>

   ...
```

Figure 8 shows how to define resources for use in an experiment. Within the `<timeline>` you define an element called `<resources>`, in which you declare a file and a randomisation for each of your resource sets. If your resource files reside in a directory other than the default experiment directory, you can specify that directory with the `dir` attribute. You should declare the path *relative* to the experiment directory. In the example the resource files will be found in `release/data/experimenter/experiment/myresourcefiles/`.

Names for resource sets Within the `<resources>` element, you define an element for each of your resource files. The name of the element is the name you will use to refer to the set when you wish to import resources from it, and can be anything you wish as long as it does not conflict with the predefined elements such as `stage`, `slide`, `timeline` and so on. The content of the element is the name of the XML file containing the resources (relative to the `dir` defined in the parent `resources` tag). In the example, we create two resource sets, sentences and artists, from the resource files `sentences.xml` and `art.xml` respectively.

Images To use image resources from your resource files, you should additionally define an `imagebase` element in the timeline, which sets the directory where the images may be found. Note that the `imagebase` must be defined relative to the WebExp2 directory rather than the experiment directory; this is to allow image resources to be used for multiple experiments, while resource files should be experiment-specific. So in the example, WebExp2 will look for the image files in `release/resources/art/`. Images can be further divided within this base directory as long as the relative path is given in the resource file, for example `<resource id="01.01">subdir/image1.jpg</resource>`. You can use JPEG, PNG or GIF images.

Importing resources To import a resource into a component in your experiment file, it is only necessary to add an `<import>` tag defining the resource set you wish to use. In the following example we import the next

resource from the set we named sentences:

```
<component>
   <type>text</type>
   <fgcolor>red</fgcolor>
   ...
   <import resource="sentences"/>
</component>
```

Any properties which are defined in the component tag, such as a red foreground colour in this example, will serve as defaults; they will be overridden by any properties specified as attributes in the imported resource.

5.4.5 Randomisation of resource sets

It is also possible to define randomisation parameters for each resource set in the `randomise` attribute. WebExp2 reads the contents of the declared resource files so that it has a pool of resource sets for your experiment, along with their block structures and randomisation requirements. Each resource set is randomised according to the requirements, and every time a resource is imported into a component, the next resource in the randomised sequence is provided.

You can randomise freely over the entire set (ignoring blocks), or within or between the blocks you define. The content of the `randomise` attribute should be one of the options `none`, `free`, `within` or `between`. You can also leave the attribute out altogether.

5.4.6 Blocked randomisation

It is possible to specify `within` and `between` together, for a randomisation of randomised blocks. You can combine these two words in whatever way is convenient, as long as the two words appear in the attribute content – for example the following are all valid ways of requesting randomisation both within and between your blocks:

- `within between`
- `within, between`
- `betweenwithin`

> Next: **Decide your stages**
> The first structural elements to place in your `Timeline` are one or more `Stage`s. Then you can define a sequence of slides for each stage.

5.5 Stage

Each `Stage` takes the properties specified in table 9. You should define a name for the stage, and either a type or a list of required properties. You can also specify an ordering or randomisation.

5.5.1 Stage properties

Table 9 shows the full list of properties you can define for the `Stage`. Most important is the `type` property; the name property is also necessary for referring to the stage subsequently.

Figure 9: Stage property tags.

Tag name	Content
name	a descriptive name for the stage; used in results and in showing progress, and should therefore be short
type	a valid stage type as defined in table 10
shows_progress	an empty tag `<shows_progress/>` indicating that slide progress (e.g. 1/5) should be shown during this stage
is_timed	an empty tag `<is_timed/>` indicating that the stage should use a timer to control the presentation of stimuli
produces_results	an empty tag `<produces_results/>` indicating that results should be recorded
order	a comma-separated list of numbers defining the order in which to show the slides — the slides are numbered, starting at 1, in the order in which they appear in the XML
randomise	an empty tag `<randomise/>` which indicates you want to randomise the sequence of slides that the stage contains
repetitions	the number of times you want to repeat the sequence of slides defined in the stage – this is optional

5.5.2 Stage behaviour

There are three behavioural properties that can be set for a stage — shows_progress, is_timed and produces_results. Figure 9 describes their purpose. These properties are set individually or by defining which type of stage you want. The properties make the stage behave in different ways, appropriate to different stages of your experiment.

5.5.3 Stage types

There are currently four types of stage, based on the options we thought would be most common. Each type defines how a particular section of the experiment (set of slides) should behave with respect to a) producing results, b) using a timer, and c) showing progress. Table 10 shows the existing types of stage, and the application of each one based on how it acts with respect to results and timing.

Figure 10: The names and properties of the available stages.

Stage name	Purpose	Behaviour		
		shows progress	is timed	produces results
subject	Collect information about the participant	F	F	T
info	Provide information	F	F	F
practise	Use timing without collecting results	T	T	F
experiment	Use timing and collect results	T	T	T

Note that both the practise and experiment types display the participant's progress during the stage. If you do not want this behaviour, or would find it easier to specify the three values yourself, you can specify them individually.

Defining your own behaviour for the stage If you want to get a different combination of behaviours, you can specify the behavioural properties individually instead of specifying a type. Any options you want to use should just be included as an empty tag — the options are listed in figure 9.

Note that if you include any of these tags in addition to a `<type>` tag, they will override any settings from the type. If no `<type>` tag is included, then the three options are set to false and overridden by whichever options you include.

Collecting Subject details The stage type `subject` is provided for you to collect information which is not directly related to the experiment. The usual application of this will be to provide a stage near the beginning of the experiment in which you request details about the participant (subject) of the experiment. This will resemble a questionnaire. Details are recorded with the results.

5.5.4 Templates

An important concept in defining your stages is that of a `template`. There are two ways of constructing the sequence of slides in a stage:

1. explicitly describe each slide
2. describe one or more template slides along with a repetitions parameter

If you use the first option, you can just write a sequence of slides which are shown in the order you describe them (unless you specify an ordering or randomisation). You can also use the `variables` feature of slides.

Using the second option is very easy – you write your slides in the same way, but this sequence is used as a template, and is duplicated a number of times. The sequence numbers for slides are automatically generated.

Repetitions For a stage where the same slide (or sequence of slides) is shown repeatedly with different contents, the stage needs only contain one slide (or a sequence), along with a `<repetitions>` tag such as `<repetitions>5</repetitions>`. The changeable components within a slide should also have an `<import>` tag which identifies a particular resource set from which resources will be imported. The sequence of one or more slides will be repeated the specified number of times, with resources being imported each time from the resource set specified in the import tag.

Note: It is up to you to ensure that your resource sets contain enough resources to fulfil all the `import`s in your repeated template. If there are too few, nothing will be imported. Additionally, if you define one of the following stage reorderings, they will be applied to the sequence of slides which are generated from your templates and resource imports. In some cases this may have no practical application – usually you will want to randomise *either* your imported resource sets, *or* the slides in your stage.

5.5.5 Hard stage orderings

The first way to specify a reordering of the slides in the experiment description file is to specify an `<order>` tag in the stage. This ordering, if valid, will be used for this stage unless overridden by a proper randomisation.

An ordering is specified as a comma-separated list of integers; for a stage with n slides, this should be a list of the numbers 1 to n. However you can also leave numbers out, to provide a subsequence of the stage's slides. Thus an ordering for a 7-slide stage could look like this: `<order>5,6,2,1, 3, 7</order>`. Note that you can use whitespace in the list if it makes it more readable.

Note: If you have generated a stage from a template, you will have to take this into consideration if you decide to define a hard ordering.

5.5.6 Automatic stage randomisation

The second way to reorder slides within a stage is to specify that you want all the slides randomised. This provides the option of randomising the order of the slides within any of the individual stages defined within an experiment. Just include the `<randomise/>` tag and the stage will be randomised. That's all there is to it.

Note: If you have generated a stage from a template, the whole generated sequence will be randomised.

Note: Whenever a stage is reordered, results are output in the order in which they were collected, and the ordering in which slides were shown is output into each stage result.

> Next: **Create your slides**
> The next structural elements to place in your timeline are one or more slides for each stage. Then you can define the components (visual and other) that go into each slide.

5.6 Slide

These are the important elements that define what goes on in the experiment, which results are collected and what advances the sequence. Each `Slide` takes the properties specified in table 11, and also has a set of `<component>` elements.

You should define a name for the slide, and a way of advancing it. Optionally you can also define a default appearance (font, background and foreground colours), focus one of the components, include a descriptive tag, and define results to reuse from previous slides.

5.6.1 Give the Slide a name

Slides and components each have two identifiers: a unique ID for programmatical reference and a name for meaningful reference. The name should be unique if it is to be used to distinguish a slide in the results.

5.6.2 Define the appearance with font and colours

You can define default font and colour settings for the whole slide, which will be used for all components.

Fonts Font elements have three sub-properties:

1. `face` is the name of a font as defined below
2. `size` is an integer point size; 14 is a reasonable starting point for informational text, while 60 will produce very large text for use as a stimulus
3. `style` is a number representing the style of the font (`0`=normal, `1`=bold, `2`=italic, `3`=both)

Note on font faces Font `<face>` tags need the proper name of a `logical font`, a `font family` (like `Helvetica`) or a `font face` (like `Helvetica Bold`). It is advisable to stick to the logical font names `Serif`, `SansSerif`, `Monospaced`, `Dialog`, and `DialogInput`, which are platform-independent. If you use the name of a font face or family, there is always the chance that it will not exist on a target system. If a font is not found at any stage, a default will be used.

Figure 11: Slide property elements/tags.

Tag name	Content
infotag	a piece of information to be associated with the slide; this is ignored during the experiment and is merely output with the results
name	a descriptive name for the slide; used in results output
seqno	the sequence number of the slide
focus	name of the component to focus
mouse_advance	an empty tag indicating that the mouse is to be monitored
advancekeys	characters that can be used to advance the slide
rows	a comma-separated list of proportional row heights in decimal format, summing to 1.0
cols	a comma-separated list of proportional column widths in decimal format, summing to 1.0
font	tags face, size, style defining the properties of the font
fgcolor	tags r, g, b defining red, green and blue components of the foreground colour; or a name defined in `java.awt.Color`
bgcolor	as above, defining the background colour
Sub-element tags for complex properties like font and colour	
r	red component of a colour (0-255)
g	green component of a colour (0-255)
b	blue component of a colour (0-255)
face	name of the font
size	integer point size for the font
style	a number representing the font style (0=normal, 1=bold, 2=italic, 3=both)

The logical font names are mapped to local system fonts on the computer running the applet and thus may still give a varying visual appearance. Despite this, the variation should be minimal if you are working within a particular social group — for example if you are running an experiment in Japanese, your subjects may be expected to have the necessary underlying fonts.

You should be aware of these potential variations, particularly if your experiment is potentially sensitive to variations in the visual environment. The only way to be sure of what your subject sees is to use an image.

Colours Precise colours can be specified by defining values between 0-255 for the sub-properties r, g, b of the `<bgcolor>` (background colour) and `<fgcolor>` (foreground colour) tags. Alternatively, some simple colours can be specified by name instead of rgb tags (see figure 12).

Figure 12: The currently available colour names.

black	white	cyan	magenta	yellow
dark_gray	gray	light_gray	pink	orange
blue	red	green	maroon	navy
teal	olive	purple	dark_red	

5.6.3 Supply a way of advancing the Slide

You must always specify sufficient components and properties to allow the participant to advance to the next slide. There are several ways to ensure the navigability of the slide sequence:

1. include one or more buttons on the slide
2. define mouse advance for the slide
3. define key advance for the slide

It is sensible to use the key and mouse advance methods exclusively from the button method, to ensure you get the response you require. Mouse clicks and key presses are primarily useful to allow a quick advance of the slide, if for example you are recording response times or want to map keys to response options. There are situations where some options will not make sense, for example key advance on a slide with text inputs.

Button advance Any button placed on the slide will advance the slide when clicked. The name of the button is recorded as the `advancer` in the results.

Mouse advance To use mouse buttons to advance the slide, include `<mouse_advance/>` in the slide element. When the mouse is clicked within the applet, the mouse is recorded as the `advancer` in the results, along with the number of the mouse button pressed. **Note:** Input components become focused when you click the mouse on them, and button components advance the slide as a button advance.

Key advance To use particular keyboard keys to advance the slide, include the key characters in an `<advancekeys>` tag in the slide element. For example, `<advancekeys> 12</advancekeys>` will make the slide advance when 1, 2 or the spacebar are pressed. The key that was pressed is recorded as the `advancer` in the results.

5.6.4 Set the focus of the Slide

It may enhance the usability of your experiment greatly to have a particular component focused when the slide is displayed. If you use an 'ok' button to advance each slide, it is helpful to have this focused so it can be clicked by pressing the return key. If you are collecting input, it is helpful to have the first input field focused so it can be typed into straight away.

To optionally set the slide's focused component, include the `<focus>` tag in the slide element. Its content should be the name of the component to focus, so if you have a component to which you have given the name `myButton`, you can focus it by including `<focus>myButton</focus>` in the slide.

5.6.5 Tag the Slide with custom information

It is possible to tag a slide with some meaningful information, for example describing the slide's 'correct' response. You can do this by placing your information between a pair of `<infotag>` tags in the slide element. This tag is output directly into the slide results.

5.6.6 Define variables

The final thing to include in a slide element is a description of any `variables` you want to use. A `variable` in this case maps a result from a component on a previous slide to the value of a property of the current slide.

Note: This function has not yet been adapted to work with templates, because of the method of referring to a previous slide as the source of the variable. If you wish to use variables, the stage in which you use them must be fully defined, without the repetitions parameter.

If you wish to use a variable, you must identify both the `source` of the variable (the result value), and the `target` for the variable (which component on this slide is to use its value).

Figure 13: Defining the source and target of a variable.

```
<variables>
   <var1>
      <source>Slide 1:letters:text</source>
      <target>
         <compname>message</compname>
         <key>content</key>
      </target>
   </var1>
</variables>
```

An example is shown in figure 13. This `variables` element specifies one variable called `var1`. Variables can be given any name except those already in use by structural elements of the experiment (you must never reuse names such as `slide`, `component` and `timeline` in any tag). Each variable must be defined within differently-named elements — we suggest incrementally-numbered names like `var1`, `var2` and so on, as the names are not used in any way.

Variable source The `<source>` tag describes where the value is to be found; in the above example the applet will attempt to get the value of the `text` result for the `letters` component on the slide named `Slide 1`. **Note:** You can only use results from slides in the current stage.

Variable target The `<target>` tag defines the property which is to be set to the value. The target must include the name of a component on the current slide in the `<compname>` tag, and the name of a property of that component in the `<key>` tag. In the above example the `content` property of the component named `message` will be set to the value retrieved from the source.

Next: **Create a layout for your components**
The next structural elements to place in your timeline are one or more components for each slide. By default your components are placed on to the slide in the order in which you specify them, and each takes up a row. You can optionally define a layout for your slide which allows more control over placement of your components.

5.7 Layout

It is possible to split a slide into rows and columns of sizes proportional to the overall slide size. Additionally the components within each of these rows or columns may be aligned. It is not possible currently to specify

alignments for individual components in cells. A sample slide layout specification is shown in figure 14; this can be used to create a layout similar to the one in figure 15.

Figure 14: Specifying a slide layout, with proportional row and column sizes, alignment and padding.

```
<slide>
   <rows>0.4T, 0.2, 0.4B</rows>
   <cols>0.3L, 0.4, 0.3R</cols>
   <vpad>50</vpad>
   <hpad>20</hpad>
   ...
</slide>
```

5.7.1 Proportional row and column sizes

Proportional sizes are specified between `<rows>` or `<cols>` tags and take the form of a comma-separated list of decimal values, as demonstrated in figure 14. The sizes should sum to 1 or the grid will run off the slide display.

Note that if row and column specifications do not match up with the number of components or do not sum to 1, the system will still attempt to use them and will give accordingly incorrect results.

5.7.2 Horizontal and vertical alignment

Alignment is specified with a single character after each decimal size. Spacing and case can be varied as required to make the parameters more readable, without breaking the experiment. Any row or column's alignment may also be omitted to get centre alignment.

Alignment characters are `tmblcr`: `top`, `middle`, `bottom` for vertical alignment within a row, and `left`, `centre`, `right` for horizontal alignment within a column. Note that the `c` and `m` parameters are interchangeable.

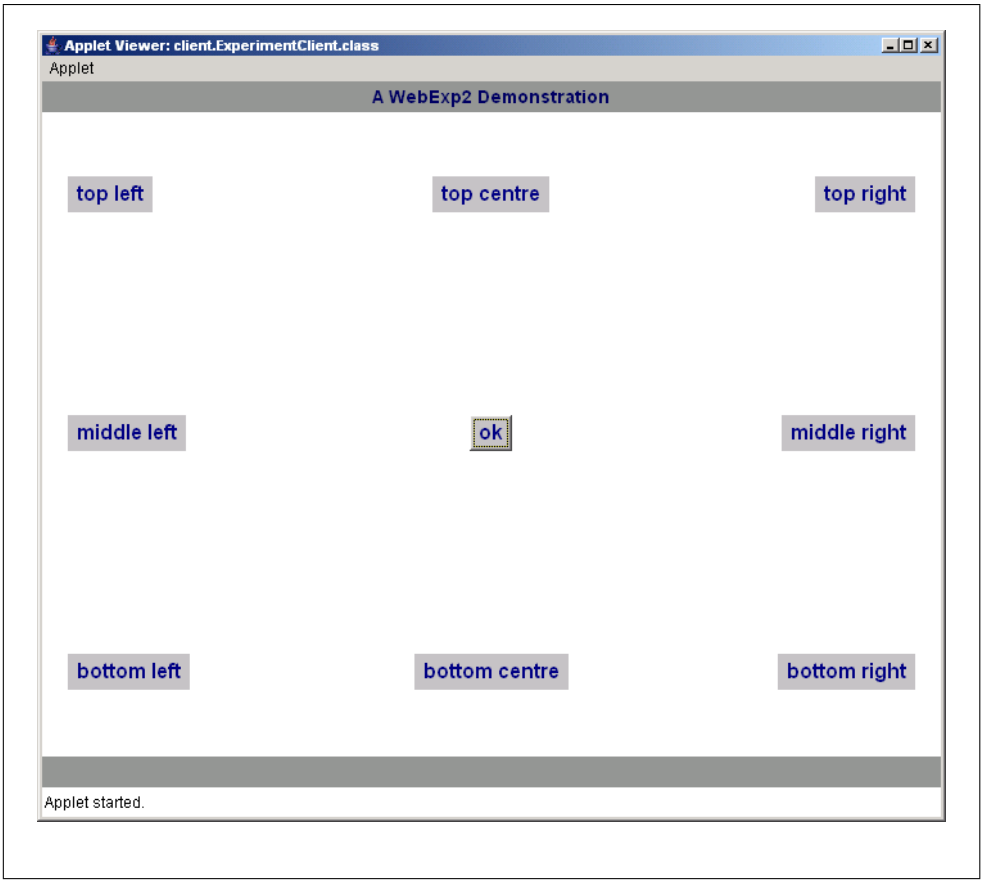
5.7.3 Padding

You can add `<hpad>` and `<vpad>` tags to a slide to define the padding (in pixels) between cells. This padding is also placed between cells and the border. Figure 14 describes a slide with 50 pixels of vertical padding between each row, and 20 pixels of horizontal padding between each column.

5.7.4 Default behaviour

If column sizes are omitted, components are laid out in a single column. If the row sizes are omitted, components are laid out sequentially in the available columns, and each row height is determined by the height of the tallest component plus padding. If any alignment is omitted, centre/middle is assumed. The demo experiment `layouttest` and its associated html file demonstrate some of the layout possibilities.

Figure 15: A sample layout split into 3 columns and 3 rows, showing padding and alignment.



5.7.5 Further advice about layouts

Note that specifying the horizontal and vertical layout parameters of rows, columns and alignment does not really create a grid of boundaries, but instead creates a framework used for determining position. So for example, components wider than a column are aligned with respect to the column but will extend outside the width of that column.

Note that visual components are placed in the layout from the top left to the bottom right, in the order in which they appear in your xml. Each component is associated with a particular 'cell' and is aligned with respect to that cell's boundaries according to the alignment values of that row and column. You may not want to place a component in every cell, and some of the following options are useful:

- For an empty cell, you can specify a completely empty component `<component/>`.
- For an empty *spacer* cell, specify a text component with no content or colours, along with height or width. A component will be created at the specified size but will be invisible.
- For a patch of colour, use a text component and specify only dimensions and background colour.

Next: **Create your components**
The next structural elements to place in your timeline are one or more components for each slide. These will be placed into the layout in the order they are specified, from the top left cell to the bottom right cell.

5.8 Components

Components are the elements you place within your slide both to create your stimulus and to collect the participant's response. Each `component` takes the basic set of properties specified in table 16.

Figure 16: Component property tags.

Tag name	Description	Required
`name`	a descriptive name for the component; used in results and variable references	Y
`type`	the type of component to create	Y
`content`	the content of the component; the meaning of this varies depending on the type	N
`id`	**(not used)** may be useful to distinguish the descriptive names from a simpler identifier, which can be used for e.g. variable references	N

You should define the `type` of component you wish to create, give it a unique `name` if you wish to refer to it unambiguously later (for example if you want to use the value of a response in a later slide), and set the supplementary properties which are relevant and necessary to this type of component, as described in table 17.

You can also import an external resource into the component, as described in section 5.4.4. The resource can fill in any properties which you have omitted.

5.8.1 Set the type

When you describe a component in your XML, you must set a property defining the `type` of component to create, for example `text` or `image`. Table 16 describes the basic properties of a component, while table 17 shows the types of component that are available along with the extra properties applicable to each type.

5.8.2 Set the relevant properties

Table 17 lists the available component types. The columns indicate a) the name that should appear between the component `<type>` property tags b) a list and description of the properties specific to each component.

5.8.3 Visual properties: font, colour, size

All visual components can have individual font and colour properties, which will override any defaults. These are specified as described in section 5.6.2. You can also specify `<width>` and `<height>` tags so that a component takes up more space or has a larger background.

Note: Although all the components currently available are visual, this will not necessarily be the case in future. However if you specify visual properties for a non-visual component, such as the colour of an audio clip, they will be ignored.

Figure 17: Available component types and their unique properties.

Type name	Property	Content
text	content	the text to show
button	content	the text for the button
image	content	the name of a preloaded image resource from the <images> preamble (see section 5.3 on loading individual images)
input	lines	how many lines to show in the input — any value greater than 1 will provide a scrollable text box
	content	**not used**; may be implemented later
	maxchars	the maximum number of characters which can be typed into the box
	allowed	a list of the characters accepted by the input
	restrict	a combination of words representing sets of allowed characters: digits, letters, symbols

Figure 18: Property tags for visual components. All are optional.

Tag name	Description
width	the width of the component in pixels
height	the height of the component in pixels
fgcolor	foreground colour to use for the component
bgcolor	background colour to use for the component
font	the font in which to display the component's text

5.8.4 Restricting input

The input response component allows the user to use the arrow keys to navigate, the delete and backspace keys to edit, the tab key to move between components, and the return key to advance the slide. Any other character key can be used to provide input, unless you restrict what the subject is allowed to type.

Figure 19 shows the definition for an input component suitable for collecting an age from the participant. It will not allow the user to advance to the next slide unless it has some input, and will only accept numbers, to a maximum of three digits. Thus effectively the user has to enter a number between $0 - 999$ in order to continue.

Limiting input length You can limit the length of your participant's input by including the <maxchars> element. Its content should be an integer defining the maximum number of characters that can be typed into the input box.

Restriction sets One way of restricting input is to include a restriction set, which defines a set of allowable characters. Add the <restrict> element, and as its content include one or more of the following:

- letters – allow alphabetical keys
- digits – allow number keys
- symbols – allow symbols to be typed

Figure 19: A mandatory `input` component for collecting a participant's age.

```
<component>
   <required/>
   <name>age</name>
   <type>input</type>
   <lines>1</lines>
   <maxchars>3</maxchars>
   <restrict>digits</restrict>
</component>
```

These parameters can be combined so for example to allow all alphanumeric characters but not symbols, you could add `<restrict>letters digits</restrict>`. The space key is accepted in addition to any restriction sets.

Custom restrictions For more control over input than restriction sets allow, the second method is to define a set of accepted characters with the `<allowed>` tag. Its content is simply all the characters that can be typed into the input box, for example `<allowed>qwerty </allowed>`. With a custom restriction, you must explicitly include the space character if you want to allow it.

Required input Response input components can be marked as `<required/>` (see figure 19) to indicate that a response is required in this component before the slide can advance. If a subject attempts to advance the slide before all required fields have suitable input, a message is displayed in the progress information bar.

5.9 Finished

Your experiment description is now complete! All that remains is to test it, write your supporting webpages and deploy it on your server.

Well-formedness If the XML is not well-formed (see 4.2.1) then you will get errors when the client attempts to read the XML from the server. You can check the XML for well-formedness with a validating XML editor; and if the client has trouble parsing your timeline, a code-colouring editor will make it easier to locate a mistake in the XML.

Next: **Test your experiment**
The next tasks are to tell the server about you and your experiment, write a page for your applet, and start the server and client.

6 Running a new experiment

This section describes how to set up a new experiment locally for testing. The basic procedures for starting the server application and the client applet where outlined in section 3 about running the sample experiments.

To tell the server about your experiment, you must update the server configuration files. You must also write a web page for your experiment, supplying an `<applet>` tag with suitable parameters.

6.1 Editing the server configuration files

You will need to edit the server configuration files to tell the server which experiments it should be administering. There is one file describing the experimenters who use the server, and each experimenter has another configuration file describing his or her experiments.

Currently you will have to restart the server if you want to add new experiments. In the future the system will be adapted so new experimenters can be added while the server is running, and their experiments can also be added or modified easily without restarting the server.

The format of these configuration files is straightforward — they are XML files and so start with the XML declaration. The body of the file is merely the root element `<config>` within which configuration information for experiments or experimenters is described. Note that XML is described in more detail in section 4.2.1.

6.1.1 Configuring experimenters

Edit the `data/experimenters.xml` file to include a record describing yourself, as in figure 20. Each experimenter needs a unique ID, and a name. The ID is used to create data directories, and to identify the experimenter when the client requests an experiment. It should not contain spaces. Usernames serve well as experimenter IDs.

Figure 20: An experimenter configuration record.

```
<experimenter>
   <id>default</id>
   <name>Dr Default</name>
</experimenter>
```

6.1.2 Configuring experiments

Create an `experiments.xml` file in your experimenter's directory (`data/yourID/`) to describe your experiments. It may be easiest to just copy the file from the `default` directory and edit the contents.

For each experiment you want to add to the server, you must add an `<experiment>` element to the configuration file, similar to that in figure 21. Each experiment needs a unique ID, and a meaningful name. The ID is used to create a data directory for the experiment, and to identify the experiment when the client makes a request.

6.1.3 Create a directory for your experiment

Ensure that a directory `data/yourID/yourExperimentID` exists, and place your experiment description file `yourExperimentID.xml` within it. Subdirectories for results will be created automatically. A running record

Figure 21: An experiment configuration record.

```
<experiment>
   <id>stroop</id>
   <name>Stroop Experiment</name>
   <max-subjects>50</max-subjects>
</experiment>
```

will also be created within each experiment's data directory, in a file with the name `yourExperimentID` (no extension). **Note:** *This file should not be deleted or edited.*

6.2 Starting the server

Start the server with the command `java -cp webexp2.jar server/Server`. This means start java, setting the classpath (`cp` – where to look for Java classes) to the WebExp2 `jar` file, and start the `Server` class in the `server` package within the `jar` file.

6.2.1 Using custom ports on your server

You can optionally specify which port to use for communication with the server. The default is port 6789, but you can use any non-reserved port which is open on your server machine. To specify port 9876, you would add `-p9876` to your `java` command. **Note:** You do not need to specify a port if you are testing your experiments locally.

6.3 Starting the client

To view the client, you have to write an HTML page with an `<applet>` tag that describes the Java applet to be displayed. This is very simple and is described in section 6.4.1. You can copy an existing sample HTML file and change the `experiment` and `experimenter` parameters.

You can then view the applet by opening your HTML page in a browser which has the Java plug-in. Alternatively, you can use a command such as `appletviewer stroop.html` at the command line, substituting the name of the relevant HTML file.

6.4 Displaying the client applet in a webpage

To make your experiment available, you must publish a webpage with the client applet embedded within it. This should typically be a page which includes some useful direction to the participant, for example a) the name of the experiment, b) an index describing keys used to advance slides.

6.4.1 The `<applet>` tag

The important HTML to include in your experiment page is summarised in figure 22. The format is simple; there is an `applet` tag which has attributes in the form attribute="`value`" like other HTML elements, and has a number of `<param>` sub-elements with their own attributes.

Figure 22: HTML to include the experimental applet in a web page.

```
<APPLET code="client.ExperimentClient.class" archive="webexp2.jar" width="700"
height="550" align="top">

   <!-- Write browser record into applet parameters: -->
   <script type="text/javascript"><!--
   document.write("<param name='browser' value='"+navigator.userAgent+"'>");
   // --></script>

   <PARAM name="experimenter" value="default"/>
   <PARAM name="experiment" value="demotest"/>
   <PARAM name="port" value="6789"/>
   <PARAM name="LANG" value="UTF-8"/>
   <alt="Java disabled!"><i>Sorry, your browser is not Java enabled.</i>

</APPLET>
```

6.4.2 Javascript and the browser

The section of the applet tag definition between `<script>` tags may appear confusing. This is a bit of javascript which will write another `<param>` tag into the HTML, describing which browser the participant is using.

The most reliable way of finding out what type of browser the participant is using is by querying the browser itself with javascript; it cannot be done in Java. This bit of code does that and writes the result to the HTML page so you get something like `<PARAM name="browser" value="Firefox"/>`. The actual value is far more lengthy, but you can usually ascertain the source browser from the name near the end of the value, or failing that the generic name near the start.

Therefore you don't need to worry about this code or how it works, just include it if you want to know what browser is being used to view your experiment. The value is added to the results and recorded in the subject results file.

6.4.3 Applet attributes and parameters

When you include the `<applet>` tag in your webpage, there are two groups of settings to configure. Firstly there are the *attributes* of the applet, which are defined in the opening tag; these are properties which relate directly to the applet. Secondly there is the set of *parameters*, which define properties which the WebExp2 client itself can read and use.

Applet attributes The attributes define basic features of the applet. They are of the form attribute="`value`" and are specified in the `<applet>` opening tag.

To centre an applet horizontally in the page, surround the `<applet>` element with `<center>` tags.

The `codebase` attribute is only necessary if the root `release` directory (with the `jar` file and data) is different from the directory containing the webpage. If your webpages are hosted outside of the `release` directory, then you should specify the `codebase` attribute. For this reason, it is easiest when testing your experiments to place their supporting HTML pages in the `release` directory and run them from there, so the codebase tag does not need to be defined. Note that the webpages may be hosted separately from WebExp2, in which case the codebase should specify the full path of the server and directory containing WebExp2.

Figure 23: Applet attributes.

Attribute	Value
code	The name of a class which contains applet code. In WebExp2 this is always client.ExperimentClient.class
archive	The name of an archive (jar file) which contains the code classes.
width	The width (in pixels) of the applet in the page
height	The height (in pixels) of the applet in the page
align	Vertical alignment of the applet within the page: top, middle or bottom
codebase	The directory containing the WebExp2 jar file, if different to the directory in which the webpage resides; this may include a full server address if the pages are hosted separately from WebExp2
name	You only need to give the applet a name if you wish to refer to it using Javascript, as described in section 7.5

Applet parameters Parameters define various values which may be used by the client; these customise the WebExp2 client applet for your use. They are sub-elements of the <applet> tag, specified in the form <PARAM name="a_name" value="a_value"/>.

We have defined these parameters within the code, and they are used to set information that is needed by the WebExp2 client to retrieve resources from the server. The important parameters are shown in table 24.

Figure 24: Applet parameters.

Parameter name	Parameter value
experimenter	ID of the experimenter whose experiment the client wants
experiment	ID of the experiment
port	The port which the server has open for communication (default 6789)
LANG	(**not implemented**) The language (character encoding) to be used in reading files from the server
browser	The browser being used to view the applet (this is set by javascript)

The experimenter and experiment parameters must be specified to identify the experiment that the client is to run. The port must match the port the server is using; if it is not set the default will be used, which is the same as the server default. The browser must be set by javascript within the webpage (see section 6.4.2).

7 Publishing an experiment on the web

The previous section explained how to get an experiment running locally for testing. This section will explain how to get your WebExp2 experiment running across the web. This involves setting up a web server, and publishing your experiment webpages.

7.1 Making experiments available on the web

To run your experiments across the web, you must make them *available* to users on the web. Ordinarily you could publish webpages on a web server which is managed for you, and indeed this is all you need to do for the pages containing the experiment client. However, the experiment clients need to contact your WebExp2 server to retrieve and return information, and you have to run this experiment server yourself and make it available via a web server. The only way to do this is to run your own web server.

7.2 Preparing for the Server

This section describes how to prepare for publishing your experiments on the web. If you are setting up WebExp2 for the first time and just want to test your experiments locally, you can skip to section 3.1.

7.2.1 Setting up a web server

You will need a `web server` in order to provide your experiments to the world via your computer. This is a program such as Apache which runs on the computer and communicates with the world wide web via a `port`, which is a virtual 'door' to the files or services you want to make available. You can restrict what is visible by configuring your web server to supply particular files only to particular groups of people.

When you have a web server such as Apache set up, you should place the WebExp2 folder somewhere in the public html tree. Ensure your `data` directory is not accessible from the web, by for example creating an Apache `.htaccess` file with the command `deny from all` in it.

7.2.2 Opening a port

You will need to ensure you have a dedicated port open on the computer where you run the WebExp2 server. This is the channel through which the server application can communicate with clients out on the web. Your web server will already use port 80 to supply HTML and other files to browsers via `http`; you will need to open a second port for WebExp2.

WebExp2 uses port 6789 by default, but any available port will do. If you use a different port, you will need to specify it when you set up experiments, and when you start the server. Note that port numbers $1024-49151$ are 'registered' and many are used for specific services; if you use one of these ports, a participant may have trouble if they happen to be using one of these services. For this reason we suggest you stick to the default of 6789, which is not currently associated with a particular service, or that you use a port in the range $49152-65535$, which are unregistered.

If you do not know how to open a port on your computer, or do not have the permissions, you should be able to make a request to your system administrator or support to open port 6789 (or another). You can explain that it is for running experiments via a client-server system.

7.2.3 Security

Only the WebExp2 applet can contact the WebExp2 server and request or return data. Conversely, the applet can contact *only* the WebExp2 server from which it is downloaded. For this reason WebExp2 itself should not pose a security risk. The server will store records it receives from the client in the data directory, which can be withdrawn from public view by placing a `.htaccess` file (for Apache) in the `data` subdirectory of the `webexp2` directory. The file should contain merely the text `deny from all`.

Opening a port on your computer does introduce a potential security risk as the port is used as a 'door' into your computer. The risk is the same as for any other open port on a computer — that it may be hijacked by a malicious program. For this reason the port should be opened on a properly firewalled computer.

Ports There is a collection of predefined port numbers which are used for particular internet services and protocols (such as port 80 for `http` connections) and which are therefore open on most servers and usable by most computers on the internet. These are 'well-known ports'.

Note that the participant's machine, which runs the WebExp2 client, must allow TCP/IP connections to lesser-known ports. This should not cause a problem in most cases, but if the client machine has a very strict firewall (as sometimes is the case in large organisations like universities), it may not allow communication with ports other than the most basic ports, such as those used for `http` and `ssh`. The best you can do is to explain this to your participants and request that they notify you of any problems.

7.2.4 Server limits

There are practical limits to the number of connections you can have to the server when it is running experiments. This is due to the safety limitations of Java, and means that in practical terms you cannot have more than about 500 people doing experiments simultaneously.

If you are expecting a very large subject pool doing experiments simultaneously, or if you are running a large number of experiments on one server, you will need to take this into account. However, for most applications you should have no need to worry.

This limitation may be removed in further development by adapting the Server application to manage connection sharing.

7.3 Setting up the system

When you have a web server set up on your machine, you should add your WebExp2 folder to the server space. You can do this by unzipping the zip file again to your server, or you might find it easier to just copy the directory in which you have been working and designing experiments. If you follow the latter course, make sure you delete any files you don't want to be visible on the web, and also ensure that each experiment directory in `data` contains **only** your experiment description file.

7.3.1 Permissions and access

You should be careful about allowing access to experimental data files (particularly results) on your web server. There are two issues here:

1. access from the internet
2. access permissions for users on your network

Firstly, data collected for experiments should not be available to browsers. It makes sense to hide your entire `data` directory on the web, so you should set up your server to refuse access to this directory and all its contents. With Apache, you achieve this with a `.htaccess` file containing the single commmand `deny from all`. A suitable `.htaccess` file is included in the WebExp2 distribution.

Secondly, you need to set access permissions for your files. It is up to you how you achieve this, but we describe how to set suitable permissions in Linux or Unix:

Your WebExp2 files and directories must be readable and executable for all users. If they aren't, then execute the following command in your `webexp2` directory: `chmod -R 755 *`.

You may want to restrict experimental data directories further. If you are the only person using the WebExp2 server to run experiments, you can restrict the data directory to yourself only, with the command `chmod -R 700 data`. If others in your network also need to read the contents of the data directory, you should issue the following command: `chmod -R 750 data`.

7.3.2 Configuration

You must configure the WebExp2 server to run your experiment. This is straightforward and you have probably already done this in testing your experiment. To add your new experiment to the server, just make sure you include an entry describing yourself in the `experimenters.xml`, and an entry for your experiment in `experiments.xml`. Setting up the server for the first time, you can of course copy the files you have been working with. If `experiments.xml` includes a reference to a non-existent experiment, that experiment will be ignored and will not be available to clients.

You should also ensure that your experiment is ready to run and collect results. If you are starting a new experiment, you should remove any running information that may have been collected during testing — if you have already run the experiment, its data directory will include results; and also information on the experiment's progress, in a file simply named after the experiment's id. The only thing you need to include to start an experiment afresh is the experiment's data directory, containing solely its XML experiment description file.

7.4 Writing supporting web pages

The final task in getting an experiment up and running on the internet is to write a web page which shows the client applet. This was described in section 6.4. However it is also very important to write some supporting webpages which will prepare the participant for the experiment, introducing relevant concepts and priming them so they undertake the experiment under conditions most conducive to getting viable results.

There are three things to consider:

1. Leading in to the experiment with some background and a description of the task ahead
2. Providing information and feedback to the participant during the experiment
3. Controlling the environment in which the experiment is undertaken

7.4.1 Leading in to the experiment

It is a good idea to lead the subject into the experiment gradually by providing supporting pages, which explain the experiment and how the subject is expected to interact with it. Then the experiment can be presented on a fresh page without background distractions. Of course it is possible to include the applet anywhere in the middle of a web page if this suits your purpose.

The best place for this information is in the supporting HTML, firstly so that you can be sure the information appears to the subject before the experiment is undertaken, and secondly so that you don't have to write it into the experiment description, which is a lengthier task!

Background The lead-in page(s) should include a number of pieces of information in order to properly prime the subject for the experiment and to try and ensure that it is done correctly:

- A description of the experiment – its history, its purpose, what it involves, time required to complete it
- Instructions concerning how the experiment will be administered – this may include reference to specific features of WebExp2 such as the navigability of the applet interface (see section 5.8.4), and a description of the input method (for example, four keys mapped to colours)
- Warnings about content – for example adult content or flashing images
- Advice about what is required of the participant – a browser with Java, a particular age or language or background
- Explanation of how to fill in the personal details questionnaire
- Reassurance about the results of the experiment – describe how security and anonymity are enforced

To get the best results, the instructions should be as clear and concise as possible, and it may be worth supplementing them with a training phase within the experiment, using the `practise` stage as described in section 5.5.3. It may also be very instructive to include advice about what **not** to do!

Ethical guidance There are ethical considerations for which solutions may need to be provided, such as content warnings, guarantees of anonymity, and so on. The official guidelines concerning experimental ethics in psychology can be found at the following links:

'Ethical Principles for Conducting Research with Human Participants' (British Psychological Society) at the rather painfully verbose address:

`http://www.bps.org.uk/the-society/ethics-rules-charter-code-of-conduct/code-of-conduct/ethical-principles-for-conducting-research-with-human-participants.cfm`

'Ethical Principles of Psychologists and Code of Conduct' (American Psychological Association) at:

`http://www.apa.org/ethics/code.html`

7.4.2 Providing information/feedback for the participant

You could consider providing some information on the page on which the applet itself appears — a contact email, a link back to a supporting page, a title, some advice about participation. Of course, depending on the type of experiment you are performing, it may well be prudent to omit these details to minimise distractions and to ensure the applet appears without requiring the page to be scrolled.

Legend A useful addition on the applet page is a legend, describing the keys which may be used for input or for advancing the slide; particularly if you want a swift response from a set of options. Including this information in the webpage is far easier than including it in each slide of the experiment, and reduces potential distractions within the applet display.

Experiment information Although information can be displayed within the title bar of the applet, you may wish to provide more detailed information describing the experiment, within the webpage. To aid this endeavour, it is possible to get certain details from the applet itself, using javascript. This is described in section 7.5.

7.4.3 Controlling the participant's environment

The best way to control the environment in which the experiment is performed is to explicitly explain your requirements to the participant. It is particularly important in web-based experimentation to try and control the context in which the experiment is administered, as there are a lot of factors which can affect your results, especially if you are recording reaction times. There are also ways to control the environment from the applet, but these are not 100% effective.

Advice to the participant You might want to provide some general instructions to ensure the veracity of your results:

- make sure the applet is fully visible in the browser window before starting
- maximise the browser window so there is no background distraction
- close down other applications to enable timing accuracy – it is no good unfortunately to have e.g. other programs downloading in the background while a subject is doing the experiment
- ensure the mouse pointer is within the applet window – if you are using key advance in your experiment and your participant is using a Mozilla browser in Linux (for example), the focus follows the mouse pointer, and the applet will not receive key presses unless it is focused

Programmatic control Two main features will aid the control of your experiment; these are not yet implemented but are high priority for future development:

- full screen display – Java allows you, in many cases, to make the applet take up the whole screen and exclude other windows, so there is no distraction; this is not 100% reliable however
- window monitoring – it is possible to be notified when the participant switches to another window, and to therefore invalidate or suspend the experiment

7.5 Javascript access to applet information

If you are familiar with Javascript, you can embed script elements in your applet webpage which can actually query the applet for information. You access public methods of the applet with code of the form `document.appletName.method`, where `appletName` is the name you defined in the `name` attribute of the applet, and `method` is the name of the method you wish to access.

Figure 25 lists the applet methods which are available to javascript. Note that all of these methods return a string. For example `document.we2app.getTitle()` will supply a string representing the title of the applet.

Unfortunately any method which returns dynamic information that changes as the experiment progresses (such as the stage name), are of limited use, as there is no way or the applet to notify javascript when such information changes.

Figure 25: Public methods of the client applet, which can be accessed by javascript.

Method	Return value
`getTitle()`	the experiment title (as defined in the experiment description file and displayed in the title bar of the applet).
`getStageName()`	the name of the current experimental stage
`getSlideNum()`	the number of the current slide
`getSubjectDetails()`	the details recorded to identify the subject's environment (see section 8.1)
`getUid()`	the unique id used to identify this instantiation of the experiment – can be used to get the results for this experiment
`getParameterInfo()`	a description of the parameters that may be used in the `<applet>` tag (**not implemented**)
`getAppletInfo()`	copyright information for the WebExp2 applet

7.6 Publicising your experiment

Once your webpages are prepared and tested, you need to upload them to a web server. This can be your own web server on which you will run the WebExp2 server application, or it can be any other web server. If you do decide to have the pages hosted on a different server, you will need to ensure that the applet tag specifies where the WebExp2 client applet is to be found (i.e. the server where WebExp2 is running). This is done using the `codebase` attribute as explained in section 6.4.3.

Once you have set up both your webpages and the WebExp2 system on web servers, you just need to start the WebExp2 server, which will then wait for connections from the WebExp2 client running in your webpages. Of course you also need to publicise your experiment. One possible host for publicising your experiment is `http://www.language-experiments.org`.

8 Results and their interpretation

Once you have collected results from your experiments, you will want to access and transform/summarise these results in some way. This section describes the format the results take and how to process them usefully.

You have three files for each successfully completed experiment, each in a separate directory:

log in a simple text format; this can be consulted to give insight into any problems that may appear in the experimental results

subject an XML file describing the subject who completed the experiment

results an XML file describing the experimental sequence that was administered, and the results that were collected

Note that each file has the same name, constructed from the id name of the experiment (the prefix), and an iteration number (the suffix). The files have different extensions however; logs have the extension `.log` while other results files are in XML format and therefore have the extension `.xml`. Thus it is easy to select matching records for a particular experiment, from the `logs`, `subjects` and `results` directories.

8.1 Guide to the Subject file

This section describes the subject results file. Here is a sample subject file:

Figure 26: An experiment configuration record.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Mon Jul 11 14:36:57 BST 2005 -->
<subject name="Subject">
   <ip>129.215.218.200</ip>
   <os>Linux 2.4.20-31.9_v1_dice_1smp i386</os>
   <jver>1.4.2_06</jver>
   <jvend>Sun Microsystems Inc.</jvend>
   <browser>Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.5)
Gecko/20041215 Firefox/1.0</browser>
</subject>
```

The browser string is taken from the 'User Agent String' which a browser provides when viewing your pages. Note that browsers are not required to provide correct information in this string and some browsers (e.g. Safari) actually identify themselves as something else. Unfortunately there is no way around this.

You will most likely come across Safari (on Apple computers), Mozilla, Firefox, MSIE, Opera, Konqueror or Netscape.

Many browsers have evolved from older browsers, whose names still appear in the browser string for compatibility. For example, Mozilla is the base for Firefox and Netscape, and Gecko is the original base for Mozilla.

For more information about browser identification strings, please view the useful resource http://javascriptkit.com/jsref/navigator.shtml — the section 'Additional browsers' Navigator Information' shows the kind of strings you will get for different browsers. The page at http://www.pgts.com.au/pgtsj/pgtsj0208b.html provides a good discussion of the problems in browser identification and how they arose.

Figure 27: Description of the parameters of a subject file.

Parameter name	Parameter value
subject	The root element; has a name attribute identifying the subject (**not used**)
ip	The ip address of the subject
os	The operating system the subject was using; while this may look complicated, the simple name `Linux` or `Windows` will usually appear at the beginning of this parameter
jver	The version of Java the subject was using to view the applet
jvend	The vendor of the Java JVM that was being used (there may be slight differences between implementations)
browser	The browser that the subject used to view the applet; this is a complicated string but the common name of the browser will usually appear at the end

8.2 Guide to the XML results format

This section describes the primary results file and the format of results you get from each of the experimental components. The results format is quite similar to the experiment description, and you will find it quite easy to read as you become increasingly familiar with XML.

The best way to get a feel for the results file is to view it in an XML editor, or a web browser, where elements are properly indented and can be collapsed or expanded, making the file more easily readable. Figure 28 shows the start of a results file for the demonstration experiment `demotest`.

The root element of the results is the `experiment`, which has a name attribute identifying the experiment. This contains a number of `stage` elements which each contain a number of `slide` elements. The relevant results of each slide are listed within these.

Note that slide and component results appear in elements which are named after the Java class which describes them, thus the names are preceded by a package name and a dot, such as `display.Slide`. This will simplify automated processing of results files in future releases, as a source file can be consulted to find out what result elements a particular component outputs.

The following sections describe the kind of result elements you will find for each component. Each slide and its components is identified by a name attribute which contains the name you specified in the experiment description.

8.2.1 Slide results

The `display.Slide` results element contains certain identifying information, along with a timing result and a record of what advanced the slide, and also results from its response components. Figure 29 describes these result elements.

8.2.2 Response Component results

(Input) The input response component provides results in a `display.TextResponse` element, containing elements described in figure 30

8.2.3 Timing results

Currently, timing results are produced with the names `inputStart`, `inputRT`, `time_shown`. Whenever a timing is output as part of the results, it consists of a `time` in milliseconds, along with a `confidence_interval` calculated with the algorithm described in Eichstaedt (2001).

8.2.4 Slide advancer

The `advancer` result consists of two sub-elements. `type` describes what caused the slide to advance and `input` describes any value recorded for the method of advancement (see figure 31).

8.3 Log files

All messages generated over the course of an experiment are saved in a log file, which is returned to the server when an experiment is successfully completed. Figure 32 lists the message prefixes which are used in log messages.

Note: Currently you will only receive logs for completed experiments. So the primary use of the log files at the moment is to verify that a completed experiment ran as you expected, without warnings or errors.

8.4 Writing webpages/scripts for parsing

8.5 Transforming results - XSL

http://www.w3schools.com/

http://www.w3schools.com/xsl/default.asp

Figure 28: An example results file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Tue Jun 21 18:14:43 BST 2005 -->

<experiment name="A WebExp2 Demonstration">
   <stage name="Subject Info">

      <randomisation>
         <ordering>4 2 5 1 3</ordering>
      </randomisation>
      <display.Slide name="Collect Subject Information">
         <advancer name="advancer">
            <type>enter</type>
            <input>age</input>
         </advancer>
         <time_shown>
            <time>32571</time>
            <confidence_interval>1628.0</confidence_interval>
         </time_shown>
         <seqno>1</seqno>
         <infotag>Some private information for the results.</infotag>
         <display.TextResponse name="name">
            <text>Neil</text>
            <inputStart>
               <time>29335</time>
               <confidence_interval>1466.0</confidence_interval>
            </inputStart>
            <inputRT>
               <time>1363</time>
               <confidence_interval>68.0</confidence_interval>
            </inputRT>
         </display.TextResponse>
         ...
      </display.Slide>

   </stage>

         ...

</experiment>
```

Figure 29: Result elements produced for a Slide.

Element name	Results content
advancer	an advancer result indicating what method was used to advance the slide
time_shown	a timing result indicating how long the slide was visible
infotag	a descriptive tag, output directly from the experiment description (see section 5.6.5)
seqno	the sequence number of the slide (**not used**)

Figure 30: Result elements produced for a text input response.

Element name	Results content
text	The text which was entered in the response component
inputStart	A timing result indicating the time that elapsed between the display of the slide and the subject's starting to enter text
inputRT	A timing result indicating the time elapsed between the subject's starting to type and the advance of the slide

Figure 31: Possible values for the type and input elements of the advancer result.

Type	Meaning	Value of input
key	A key caused the advance	Key character
button	A button component caused the advance	Button label
mouse	A mouse button caused the advance	Mouse button number
enter	The enter/return key was pressed in an input field, causing the advance	Name of the input component in which return was pressed

Figure 32: Prefixes to messages which are displayed in the console and written to log files.

Prefix	Purpose
WARNING	A warning message about potential (non-fatal) problems or unexpected behaviour – for example when default values are assumed.
ADVICE	Advice about good experimental design practice.
ERROR	Information about errors, which may or may not have been fatal.
INFO	Generic information which may be useful.
DEBUG	Debugging information – switched off in the release version of WebExp2.

9 Troubleshooting

If you have trouble, there is usually plenty of information giving you hints about what might be wrong, or at least where the problem is. Java will tell you of any 'runtime errors', and WebExp2 will let you know if anything goes wrongly or unexpectedly within the program.

The first place to look if you have trouble (for example, if the experiment won't start, or it fails) is at the console output of the system. Messages are generated both to report errors and to describe the progress of an experiment, and these messages are output to a console for both the server and clients. Ultimately this information is logged to a file which is available to the server. Section 8.3 describes the type of messages that are generated.

9.1 Client messages

Client output goes to the Java console, which is not displayed by default but may usually be opened either from the browser (although note that Firefox for some reason has failed to include this option) or from an icon (in the system tray in Windows). If a participant reports problems you may be able to get them to open the Java console and see what output has been generated describing the problem.

Note: Currently you will only receive logs for completed experiments. Eventually output for uncompleted experiments will also be sent back to the server so it will be easier to analyse problems.

9.2 Server messages

Server output goes directly to the console in which you run the server, so you can check this at any time for problems. Due to the limited size of a console buffer however, older messages will eventually be lost as new ones are generated.

Note: The server is prevented from saving its own log files at the moment, until the system can be extended to manage the increasing size of a log file while a server continues to run. Messages are still generated and printed to the console however.

9.3 Common problems

This section describes problems which may crop up more regularly, and which have simple solutions — they may be due to a problem with your server setup or an error in the experiment description. This list will be expanded

Mismatched ports Make sure the port specified in your applet parameter is the same as the one used when starting your server.

9.4 Experiment/Client problems

If you have problems running an experiment at the client end (i.e. in a browser) then you can first open the Java console (from a browser menu, or via a system tray icon in Windows) and see if it reports any runtime errors. It should also give you feedback while the experiment is running. This information is logged and sent back to the server at the end of an experiment.

Client not contacting the server When the applet loads it attempts to contact the server. If for some reason the client can't be loaded or the server cannot be contacted, the server knows nothing about it. For this reason you will know nothing about people who could not access the applet.

You could also ask people to report if they are having problems connecting.

9.5 Bugs

There are inevitably some bugs in the current release of WebExp2. We are aware of several and will be working on them; if you find any significant bugs while using the system, please report them to webexp-dev@inf.ed.ac.uk.

10 Glossary

Figure 33: Technical terminology used in this document.

Client	An applet obtained from the Web Server by a browser viewing html pages. The applet runs in the browser and communicates with the Server.
Server	An application which can be contacted for the exchange of information. This can be a web server (which serves up HTML pages) or a different type of server *which runs* **on** *a web server* and serves up different types of information.
Web Server	A machine which hosts web pages and applications, and 'serves' them to clients.
Port	A 'door' on a server which is used to provide a particular service; clients on the web conenct with the server through its port
XML	eXtendend Markup Language; a language for marking up or 'tagging' sections of textual documents with a meaning.
Java	An object-oriented (OO) programming language allowing cross-platform compatibility and web applications.
JVM	Java Virtual Machine; the environment in which java runs its programs. This is what allows Java to run on different platforms – your JVM is specific to your OS.
JRE	Java Runtime Environment; a Java environment for your computer which includes a JVM and enables you to run Java programs (locally) or applets (locally or in your browser).
Java Plug-in	A simple Java environment which provides a JVM purely for running applets in your browser. This is all your subjects will need.

Table 34 explains some of the terminology used in describing experiments. Though familiar, these terms may well be used in a different sense to that which you understand. Please refer to this table in relation to experiment descriptions.

Figure 34: Terminology used in describing Experiments.

Timeline	The description of the whole sequence of events in an experiment, from which the full dynamic experiment is produced.
Stage	A section of the experiment with a particular aim, such as collecting subject details, providing practise, or administering the experiment proper.
Slide	A single display within the Timeline of the experiment; it contains components (stimuli and responses/inputs). You will tend to use one Slide per condition.
Component	Something which can be *presented* to the participant in a Slide. This includes both Stimulus components, which are presented to provoke a response in the subject; and Response components, which are presented in order to collect input representing the subject's response.
Stimulus	A component which simply presents something to the participant, like some text, a picture or a sound.
Response	A component to which a participant can *respond* by providing input, in the form of text or a choice.

11 Copyright

Some information on freedom of usage, and freedom of source code.

Copyright details are being finalised.

11.1 External code

WebExp2 uses a modified version of a utility class `DOMUtil.java`, written by Sun Microsystems, which requires the following copyright notice:

12 Contact details

The current contact address for the developers is:

Martin Corley
Department of Psychology
University of Edinburgh
7 George Square
Edinburgh EH8 9JZ, UK

Please subscribe to the following mailing list if you would like to receive announcements concerning the WebExp2 classes:

webexp@cogsci.ed.ac.uk

To subscribe to this list, please send a message containing the word '`subscribe`' in the message body to:

webexp-request@cogsci.ed.ac.uk

If you would like to obtain the latest version of the classes (or the source code, in case you want to participate in the development), please contact the following address:

webexp-dev@cogsci.ed.ac.uk

This address should also be used for any other queries regarding the classes, including bug reports.

References

Bard, Ellen Gurman, Dan Robertson, and Antonella Sorace. 1996. Magnitude Estimation of Linguistic Acceptability. *Language* 72(1): 32–68.

Cowart, Wayne. 1997. *Experimental Syntax: Applying Objective Methods to Sentence Judgments*. Thousand Oaks, CA: Sage Publications.

Eichstaedt, Jan. 2001. An inaccurate-timing filter for reaction time measurement by Java applets implementing Internet-based experiments. *Behaviour Research Methods, Instruments, & Computers* 33(2): 179–186.

Hewson, Claire M., Dianna Laurent, and Carl M. Vogel. 1996. Proper Methodologies for Psychological and Sociological Studies Conducted via the Internet. *Behavior Research Methods, Instruments, and Computers* 28: 186–191.

Larman, Craig. 1998 *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design*. Prentice Hall PTR.

Oppenheim, A.N., 1966. *Questionnaire Design and Attitude Measurement*. Heinemann.

Reips, Ulf-Dietrich. 2002 Standards for Internet-based experimenting. *Experimental Psychology* 49(4): 243–256.