

Inquisit 3 Help

Inquisit is a general purpose experimentation engine for designing and administering psychological experiments and measures. Inquisit can be used to collect a wide range of psychological data, including reaction time tasks, psychophysiological experiments, attitude measures, market research, and subjective questionnaires.

An Inquisit experiment is specified using Inquisit's powerful and intuitive scripting language. The script defines all of the pieces of the experiment such as the stimuli, questionnaire items, trials, blocks of trials, instructions, as well as the logic determining the flow of events. An Inquisit script is saved as a plain text file with the *.exp file extension. The script file can be opened, edited, and run using the Inquisit editor.

The Inquisit scripting language consists of two simple syntactic constructions - *elements* and *attributes*. Elements correspond to the things that make up a script, including [surveys](#), survey pages, experiments, blocks, trials, stimuli, instruction pages, the data file, and so forth. Each element, in turn, has a set of attributes that determine exactly how that element behaves. For example, Inquisit's text element, which defines a set a text stimuli, has attributes that specify the color, font, and screen location to present the text.

The Inquisit language was designed to be easy and approachable to nonprogrammers who are familiar with the basics of experimental psychology. The structure of the Inquisit scripting language should be familiar to anyone who has edited HTML. However, some of the cumbersome aspects of HTML syntax have been streamlined to make Inquisit scripts easier to read and write. Writing an Inquisit script is simply a matter of defining the elements of your experiment and setting their attributes to the desired values. Once you understand this basic idea, it's just a matter of familiarizing yourself with the details of the elements and attributes. To get started learning how to write Inquisit scripts, read through the tutorials.

Inquisit scripts are written and run using the Inquisit editor. Once you have written an experiment, you issue commands from Inquisit's menu to run parts of the experiment or the whole thing. When a script is run, Inquisit first parses its commands. If the script contains no errors, it runs. Otherwise, Inquisit reports the errors in the righthand panel of the editor. Note that Inquisit does not compile a script into an executable file that can be run by itself. Running a script thus requires that the Inquisit program or web control be installed on the machine

As Inquisit Desktop Edition runs an experiment, it writes the data to a file. By default, the data file is located in the same folder as the script file using the same name as the script file except that the file extension is changed to "dat". By default, Inquisit Web Edition saves data files to the Inquisit web server where you can log in and download the files.

For reaction time tasks, each line of data in the file corresponds to a single trial. For questionnaires, all responses from each participant are stored on a single line. Inquisit data files can be imported directly into programs like SPSS and Excel for analysis. Inquisit can be configured to record

metrics such as mean and median response latencies, standard deviations, percentage of correct response, and even custom statistics, but for most purposes, it is necessary to analyze the data using a statistical analysis program.

To learn Inquisit, we suggest the following:

1. Read through the tutorials.
2. Download and run some of the sample experiments.
3. Make minor modifications to a sample using the language reference as your guide (tip: from the editor, press F1 and the reference topic for the currently selected element will open). Observe your modifications using the Object Browser.
4. Read through the How To section.

Inquisit Tutorials

The following tutorials will guide you through the process of implementing a variety of different data collection procedures.

Creating an experiment or survey is a simple matter of editing script containing instructions and configuration settings telling Inquisit what to do, when to do it, and which data to record (e.g. reaction times, survey responses).

The text that Inquisit uses to perform a set of operations is called a script, typically saved as a file with the *.exp file extension. If you installed Inquisit using the default options, the scripts for the tutorials were installed under the *C:\Program Files\Millisecond Software\Inquisit 3\tutorials* directory.

Simplified Implicit Attitude Task This is a stripped down nonstandard version of the IAT that illustrates basic Inquisit programming concepts. The script for this tutorial is in the file *iat_tutorial.exp*.

Standard IAT This shows how to modify a standard IAT using your own attribute and target categories. The script for this tutorial is in the file *Standard IAT.exp*.

Demographic Survey The script for this tutorial is in the file *demographics.exp*.

Standard Picture IAT This builds a standard picture IAT from scratch. The tutorial is in the file *pictureiat.exp*.

Subliminal Priming Task The script for this tutorial is in the file *subliminal_tutorial.exp*.

Covert Attention Task The script for this tutorial is in the file *ca_tutorial.exp*.

Dot probe Task The script for this tutorial is in the file *ca_tutorial.exp*.

Tutorial: Standard Implicit Attitude Task

This tutorial demonstrates how to create a standard Implicit Attitude Task (IAT) as developed by Tony Greenwald et al. The tutorial starts with an IAT that measures implicit attitudes towards flowers and insects, and shows how to adapt the task to measure other target categories.

On the following pages, Inquisit commands are printed in blue, and comments are printed in black:

Steps

1. [Modifying Attribute Categories](#)
2. [Modifying Target Categories](#)
3. [Modifying Task Instructions](#)
4. [Changing Response Keys](#)

1-Modifying Attribute Categories

The first step in creating your custom IAT is to open the sample IAT task that ships with Inquisit. Open the sample by clicking the Window Start menu and selecting All Programs->Inquisit 3->Tutorial Scripts->Standard IAT. The Standard IAT measures attitudes towards flowers versus insects. In this tutorial, we will adapt the IAT to measure attitudes towards other target categories (we'll use the example of men versus women, but you can just as easily substitute any category of interest).

The Standard IAT script was organized so that the portions to be modified are conveniently located at the top of the document. The first thing you'll see at the top is a comment with helpful instructions on modifying the stimuli:

This sample IAT can be easily adapted to different target categories and attributes. To change the categories, you need only change the stimulus items and labels immediately below this line.

This text is simply a comment for someone authoring a script. It is not a part of the IAT itself, and does not have any impact on how the script runs.

The next section of the script shows attribute labels along with the stimulus items that serve as examples of either attribute category:

```
<item attributeAlabel>  
/1 = "Good"  
</item>
```

```
<item attributeA>  
/1 = "Marvelous"  
/2 = "Superb"  
/3 = "Pleasure"  
/4 = "Beautiful"  
/5 = "Joyful"  
/6 = "Glorious"  
/7 = "Lovely"  
/8 = "Wonderful"  
</item>
```

```
<item attributeBlabel>  
/1 = "Bad"  
</item>
```

```
<item attributeB>  
/1 = "Tragic"  
/2 = "Horrible"  
/3 = "Agony"  
/4 = "Painful"  
/5 = "Terrible"  
/6 = "Awful"  
/7 = "Humiliate"  
/8 = "Nasty"  
</item>
```

For our IAT, we will change the attributes labels to "Strong" and "Weak" in order to measure how attitudes towards men and women conform to gender stereotyping, and we'll change the stimulus items to examples of these categories.

```
<item attributeAlabel>  
/1 = "Strong"  
</item>
```

```
<item attributeA>  
/1 = "Power"  
/2 = "Command"  
/3 = "Dominant"  
/4 = "Succeed"  
/5 = "Assert"  
/6 = "Confident"  
/7 = "Control"  
/8 = "Bold"  
</item>
```

```
<item attributeBlabel>  
/1 = "Weak"  
</item>
```

```
<item attributeB>  
/1 = "Timid"  
/2 = "Submissive"  
/3 = "Fragile"  
/4 = "Follow"  
/5 = "Fail"  
/6 = "Obey"  
/7 = "Hesitant"  
/8 = "Uncertain"  
</item>
```

There, that was pretty easy. At this point, we have an IAT that measures implicit associations of flowers and insects as weak or strong. While this may be of interest to botanists and entomologists, the next step for us is to modify the target categories to measure attitudes towards men and women.

2-Modifying Target Categories

The next section of the IAT script contains the definitions of the target category labels and stimulus items. The syntax is identical to the attribute definitions, only the items and item names are different:

```
<item targetAlabel>  
/1 = "Flowers"  
</item>
```

```
<item targetA>  
/1 = "Orchid"  
/2 = "Lily"  
/3 = "Violet"  
/4 = "Daisy"  
/5 = "Tulip"  
/6 = "Poppy"  
/7 = "Daffodil"  
/8 = "Lilac"  
</item>
```

```
<item targetBlabel>  
/1 = "Insects"  
</item>
```

```
<item targetB>  
/1 = "Ant"  
/2 = "Locust"  
/3 = "Bee"  
/4 = "Wasp"  
/5 = "Beetle"  
/6 = "Termite"  
/7 = "Roach"  
/8 = "Moth"  
</item>
```

The labels are defined in <item targetAlabel> and <item targetBlabel>, and the stimulus items are defined in <item targetA> and <item targetB>. First, we'll change our target labels and stimuli for category A, men, as follows:

```
<item attributeAlabel>  
/1 = "Men"  
</item>
```

```
<item attributeA>  
/1 = "James"  
/2 = "Robert"  
/3 = "Steve"
```

```
/4 = "Henry"  
/5 = "Tony"  
/6 = "Sean"  
/7 = "John"  
/8 = "William"
```

```
</item>
```

Next we'll create change the label and stimuli for category B, women:

```
<item attributeBlabel>  
/1 = "Women"  
</item>
```

```
<item attributeB>  
/1 = "Debbie"  
/2 = "Shelly"  
/3 = "Karen"  
/4 = "Anya"  
/5 = "Susan"  
/6 = "Wendy"  
/7 = "Michelle"  
/8 = "Jane"  
</item>
```

If you're goal is to simply to adapt the IAT to a particular set of attributes and targets, I have good news. You're done! To run your IAT, just select the Run command on Inquisit's Experiment menu. Now go collect some data.

In the next section, we'll cover where task instructions for in the IAT are defined in case you need to modify the text or translate the instructions to another language.

3-Modifying Task Instructions

The IAT sample includes general instructions that have been written to apply to any standard IAT, regardless of the attributes or targets. However, if you are administering the IAT to a population that doesn't speak English, or that would benefit from additional or specially worded instructions, you can easily modify the text to suit your needs.

If you scroll down below the target category definitions, you'll find the following element, which specifies the text for 7 pages of instructions that appear throughout the IAT. Instructions are defined using the <item> element, just as the attribute and target stimuli were. One difference, however, is that the instruction items formatted with line breaks, which helps make them more readable when presented on the screen.

<item instructions>

/ 1 = "Put your middle or index fingers on the E and I keys of your keyboard. Words representing the categories at the top will appear one-by-one in the middle of the screen. When the item belongs to a category on the left, press the E key; when the item belongs to a category on the right, press the I key. Items belong to only one category. If you make an error, an X will appear - fix the error by hitting the other key.

This is a timed sorting task. GO AS FAST AS YOU CAN while making as few mistakes as possible. Going too slow or making too many errors will result in an uninterpretable score. This task will take about 5 minutes to complete."

/ 2 = "See above, the categories have changed. The items for sorting have changed as well. The rules, however, are the same.

When the item belongs to a category on the left, press the E key; when the item belongs to a category on the right, press the I key. Items belong to only one category. An X appears after an error - fix the error by hitting the other key. GO AS FAST AS YOU CAN."

/ 3 = "See above, the four categories you saw separately now appear together. Remember, each item belongs to only one group. For example, if the categories flower and good appeared on the separate sides above - words meaning flower would go in the flower category, not the good category.

The green and white labels and items may help to identify the appropriate category. Use the E and I keys to categorize items into four groups left and right, and correct errors by hitting the other key."

/ 4 = "Sort the same four categories again. Remember to go as fast as you can while making as few mistakes as possible.

The green and white labels and items may help to identify the appropriate category. Use the E and I keys to categorize items into the four groups left and right, and correct errors by hitting the other key."

/ 5 = "Notice above, there are only two categories and they have switched positions. The concept that was previously on the left is now on the right, and the concept that was on the right is now on the left. Practice this new configuration.

Use the E and I keys to categorize items left and right, and correct errors by hitting the other key."
/ 6 = "See above, the four categories now appear together in a new configuration. Remember, each item belongs to only one group.

The green and white labels and items may help to identify the appropriate category. Use the E and I keys to categorize items into the four groups left and right, and correct errors by hitting the other key."

/ 7 = "Sort the same four categories again. Remember to go as fast as you can while making as few mistakes as possible.

The green and white labels and items may help to identify the appropriate category. Use the E and I keys to categorize items into the four groups left and right, and correct errors by hitting the other key."

</item>

You can modify the instruction pages just as you did the attributes and category items, by simply editing the text within quotation marks.

If you make significant change to the instructions, you might also need to change the size and location at which the text is presented on the screen. The element that controls how instruction text is presented is the following:

```
<text instructions>  
/ items = instructions  
/ hjustify = left  
/ size = (90%, 60%)  
/ position = (50%, 85%)  
/ valign = bottom  
/ select = instructions  
</text>
```

The <text> element is used to define how a set of text items are presented. The first command in the element, / items = instructions, specifies that the actual text items are defined in an <item> element called "instructions", which is the <item instruction> element we just covered above. The / hjustify = left command specifies that text should be left justified. You can also set this to "center" or "right" (e.e., for right to left languages like Hebrew). The / size = (90%, 60%) specifies the size of the bounding rectangle in which the text should be presented (and words are wrapped). In this case, the width is 90% of the screen, and the height is 60%.

By using percentages, the size will scale across monitors with different display resolutions. The /position = (50%, 85%) command specifies the position on the screen, which is located horizontally at the midpoint of the screen and 85% of the way vertically towards the bottom. The /valign = bottom command specifies that the bottom of the bounding rectangle containing the text should align with the point on the screen specified by the position command. The rest we'll ignore for now.

Note that if you need to resize the text itself, you can do so using the /fontstyle command. To change the fontstyle, insert the cursor inside the <text instructions> element and select the "Font Wizard" command from the Tools menu. This will launch a graphical font picker that will allow you to choose the font parameters and insert the corresponding /fontstyle command into the script.

4-Changing Response Keys

By default, the IAT uses the E and I keys to indicate a left and right response. The response keys are defined using the <trial> element, which is responsible for presenting stimuli and gathering responses. The IAT has six different <trial> elements. The first two, shown below, present stimuli from attribute A and attribute B respectively:

```
<trial attributeA>  
/ validresponse = ("E", "I")  
/ correctresponse = ("E")  
/ stimulusframes = [1 = attributeA]  
/ posttrialpause = 250  
</trial>
```

```
<trial attributeB>  
/ validresponse = ("E", "I")  
/ correctresponse = ("I")  
/ stimulusframes = [1 = attributeB]  
/ posttrialpause = 250  
</trial>
```

The validresponse command defines two permissible responses for either trial, "E" and "I". The correctresponse defines which of these responses is considered correct for purposes of scoring and error feedback. For attribute A, the "E" key is always correct. For attribute B, the "I" key is correct. You can change the keys used for responding by replacing "E" and "I" with whatever character you like. If you are using another input device such as a response box, you can replace these with the numeric values corresponding to the buttons.

The remaining four trials present targets A and B. Recall that the IAT presents the targets with both compatible and incompatible attribute pairings, so there are two trials defined for each target category that vary in whether "E" or "I" is considered the correct response. Again, the particular keys used can be modified as they were with the attribute trials.

```
<trial targetBleft>  
/ validresponse = ("E", "I")  
/ correctresponse = ("E")  
/ stimulusframes = [1 = targetB]  
/ posttrialpause = 250  
</trial>
```

```
<trial targetBright>  
/ validresponse = ("E", "I")  
/ correctresponse = ("I")  
/ stimulusframes = [1 = targetB]  
/ posttrialpause = 250  
</trial>
```

```
<trial targetAleft>  
/ validresponse = ("E", "I")  
/ correctresponse = ("E")  
/ stimulusframes = [1 = targetA]  
/ posttrialpause = 250  
</trial>
```

```
<trial targetAright>  
/ validresponse = ("E", "I")  
/ correctresponse = ("I")  
/ stimulusframes = [1 = targetA]  
/ posttrialpause = 250  
</trial>
```

The tutorial is now complete! You can run the IAT by selecting the "Run" command on the "Experiment" menu.

Tutorial: Standard Picture IAT

This tutorial builds an Implicit Attitude Task (IAT) modeled after the version that runs on the Project Implicit web site (www.projectimplicit.org). This version of the task measures implicit attitudes towards flowers and insects as represented by pictures.

This tutorial walks through the process of building an IAT from scratch. If your goal is simply to adapt the IAT to your own attribute and target categories, you need only make a few simple modifications to the sample script as indicated at the end of the section on [creating text stimuli](#); you can skip the rest. This tutorial will be of interest to anyone interested in the details of the IAT procedure, or who wishes to modify the procedure. The tutorial also illustrates a number of intermediate and advanced concepts that are relevant to other procedures besides the IAT.

On the following pages, Inquisit commands are printed in blue, and comments are printed in black:

Steps

1. [Creating Text and Picture Stimuli](#)
2. [Creating Instructions](#)
3. [Creating Trials](#)
4. [Creating Blocks](#)
5. [Creating an Experiment](#)

1-Creating Text and Picture Stimuli

The first step in creating our script is to define the various stimuli that will be presented in the IAT. In this case, the stimuli will be a mix of text (good and bad words) and pictures (flowers and insects). Other stimuli include task instructions, a big red "X" for an error message, and category labels to remind participants which response keys map to which categories.

The good and bad text stimuli are defined as follows:

```
<text attributeA>  
/ items = attributeA  
/ txcolor = (0, 255, 0)  
</text>
```

```
<text attributeB>  
/ items = attributeB  
/ txcolor = (0, 255, 0)  
</text>
```

The *txcolor* attribute sets the red, green, and blue components of the text color. Green is set to the maximum value, whereas red and blue are 0, so both sets of words will appear green. The actual words in the set are specified separately in item elements called "attributeA" and "attributeB", which are defined below. We'll use generic names like "attributeA" throughout the script to illustrate that the script contains mostly generic IAT logic that can be easily adapted to different target categories and attributes.

Next, we'll define the pictures representing the target categories, flowers and insects:

```
<picture targetB>
/ items = targetB
/ size = (20%, 20%)
</picture>
```

```
<picture targetA>
/ items = targetA
/ size = (20%, 20%)
</picture>
```

Like the text elements, the items are defined in separate item elements. The size of the pictures is set to 20% of the height and width of the screen. In fact, all sizes for pictures and text in this script will be defined in terms of percentage of the screen. This allows the IAT to scale proportionally to different monitor sizes, making it suitable for the web where users run a wide range of display systems.

Now we'll define the category labels that appear in the upper left and right corners of the screen. The attribute labels are as follows:

```
<text attributeAleft>
/ items = attributeAlabel
/ valign = top
/ halign = left
/ position = (5%, 5%)
/ txcolor = (0, 255, 0)
</text>
```

```
<text attributeBright>
/ items = attributeBlabel
/ valign = top
/ halign = right
/ position = (95%, 5%)
/ txcolor = (0, 255, 0)
</text>
```

The first label is presented in the upper left corner at a 5% margin from the upper and left edges of the screen as defined by the *position* attribute. The second label is presented in the upper right corner of the screen, again with 5% margins. The color of both labels is green, just like the stimuli themselves. The *items* attribute specifies that the actual label text is defined in an item elements below.

Now for the target labels. These are similar to the attribute labels, except that they are presented in the default text color (we'll set the default to white later in the tutorial).

```
<text targetAleft>
/ items = targetAlabel
/ valign = top
/ halign = left
/ position = (5%, 5%)
</text>
```

```
<text targetBright>
/ items = targetBlabel
/ valign = top
/ halign = right
/ position = (95%, 5%)
</text>
```

Recall that in the IAT, the target categories switch sides midway through the test, so we'll also define labels that place category A on the right and B on the left:

```
<text targetBleft>
/ items = targetBlabel
/ valign = top
/ halign = left
/ position = (5%, 5%)
</text>
```

```
<text targetAright>
/ items = targetAlabel
/ valign = top
/ halign = right
/ position = (95%, 5%)
</text>
```

Next we'll create category labels for the critical trials where the targets and attributes are mixed together, for example, "Flowers or Good" and "Insects or Bad". On these trials, we'll present the target labels we created above. Just below those labels we'll present the word "or", and just below those the attribute labels will be presented:

```
<text orleft>
/ items = ("or")
/ valign = top
/ halign = left
/ position = (5%, 12%)
</text>
```

```
<text attributeAleftmixed>
/ items = attributeAlabel
/ valign = top
/ halign = left
/ position = (5%, 19%)
/ txcolor = (0, 255, 0)
</text>
```

These labels are similar to those above, except that the *position* attribute specifies that the "or" label be presented 12% of the way from the top of the screen just below the target label. The attribute label appears 19% of the way down the screen below the "or". Now we'll create the corresponding labels for the right side of the screen:

```
<text orright>
/ items = ("or")
/ valign = top
/ halign = right
/ position = (95%, 12%)
</text>
```

```
<text attributeBrightmixed>
/ items = attributeBlabel
/ valign = top
/ halign = right
/ position = (95%, 19%)
/ txcolor = (0, 255, 0)
</text>
```

That does it for the labels. Now we'll define instruction text:

```
<text instructions>
/ items = instructions
/ hjustify = left
/ size = (90%, 60%)
/ position = (50%, 85%)
/ valign = bottom
/ select = instructions
/ fontstyle = ("Arial", 3.5%)
</text>
```


The instructions text element differs from the others above in that it uses the *size* attribute to define a rectangle within which the presented text is word-wrapped. This is useful for displaying sentences and paragraphs. The *hjustify* command specifies that text should be left justified within this rectangle. Whereas the previous text elements use the default font, the instructions element specifies the "Arial" font at 3.5% of the screen height.

The final difference to note is the *select* attribute. There are a total of 7 instruction items (defined below). Each time this text element is presented, a different item is selected for presentation. The *select* attribute specifies that the rules for selecting the next item are contained in a counter element named instructions, which is defined as follows:

```
<counter instructions>
/ resetinterval = 20
/ select = sequence(1, 2, 3, 4, 5, 6, 7)
</counter>
```

The counter element allows you to create a customized selection algorithm. In this case, the counter's *select* attribute specifies that items should be selected in sequence (1, 2, 3, 4, 5, 6, 7). What does the *resetinterval* mean? By default, counters only remember selection for the duration of a single block. Once that block is over, the counter is reset, its memory erased, so that next time it is used for selection, it will start the sequence from the beginning. The *resetinterval* attribute specifies how many blocks the counter memory lasts. We want the counter to track the state of the sequence for the duration of the experiment, so we've set it here to an arbitrary high number of 20. Any number greater than the number of blocks in the IAT would do the trick here.

Next we'll create a simple text that tells participants to hit the space bar to advance past the instructions. This appears in the lower middle of the screen beneath the instructions text.

```
<text spacebar>
/ items = ("Press the SPACE BAR to begin.")
/ position = (50%, 95%)
/ valign = bottom
/ fontstyle = ("Arial", 3.5%)
</text>
```

Just one last text element to create and we'll move on to the actual item sets used by some of the stimuli above.

```
<text error>
/ position = (50%, 75%)
/ items = ("X")
/ color = (255, 0, 0)
/ fontstyle = ("Arial", 10%, true)
</text>
```

The element above creates an error stimulus, which is a big red "X", presented in a bold Arial font that is 10% of the height of the screen. Hard for our participants to miss that.

If your goal is simply to adapt the IAT to a specific set of target and attribute categories, the last section below is by far the most interesting. This section defines the labels and members of each category. The logic contained in the rest of the script is generic to any category. Thus, in order to change the categories, we can make all of our modifications here and leave the rest of the IAT procedure as is.

First we'll define the label and members of the "Good" category:

```
<item attributeAlabel>  
/1 = "Good"  
</item>
```

```
<item attributeA>  
/1 = "Marvelous"  
/2 = "Superb"  
/3 = "Pleasure"  
/4 = "Beautiful"  
/5 = "Joyful"  
/6 = "Glorious"  
/7 = "Lovely"  
/8 = "Wonderful"  
</item>
```

Next we'll define the label and members of the "Bad" category:

```
<item attributeBlabel>  
/1 = "Bad"  
</item>
```

```
<item attributeB>  
/1 = "Tragic"  
/2 = "Horrible"  
/3 = "Agony"  
/4 = "Painful"  
/5 = "Terrible"  
/6 = "Awful"  
/7 = "Humiliate"  
/8 = "Nasty"  
</item>
```

Our IAT will test participants' preferences for flowers or insects, so the following specify the labels and pictures files for these categories:

```
<item targetAlabel>  
/1 = "Flowers"  
</item>
```

```
<item targetA>  
/1 = "flower1.jpg"  
/2 = "flower2.jpg"  
/3 = "flower3.jpg"  
/4 = "flower4.jpg"  
/5 = "flower5.jpg"  
/6 = "flower6.jpg"  
/7 = "flower7.jpg"  
/8 = "flower8.jpg"  
</item>
```

```
<item targetBlabel>  
/1 = "Insects"  
</item>
```

```
<item targetB>  
/1 = "insect1.jpg"  
/2 = "insect2.jpg"  
/3 = "insect3.jpg"  
/4 = "insect4.jpg"  
/5 = "insect5.jpg"  
/6 = "insect6.jpg"  
/7 = "insect7.jpg"  
/8 = "insect8.jpg"  
</item>
```

That's does it for stimuli. This section has demonstrated a number of concepts, including presenting pictures and text, controlling stimulus size and position on the screen, specifying size, color, and face of a font, using a custom algorithm for stimulus item selection, and more.

2-Creating Instructions

Instructions can be presented as text stimuli like those created in the previous section. Inquisit also provides a built-in facility for presenting instruction pages in HTML or plain text. In this tutorial, we will use this facility to present a summary of the participant's performance.

The summary page is defined as follows:

```
<page summary>
^Below is a summary of your average response time for two different configurations.
^^Configuration 1: <% item.targetAlabel.1 %> with <% item.attributeAlabel.1 %>, <%
item.targetBlabel.1 %> with <% item.attributeBlabel.1 %>
^ <%block.compatibletest.meanlatency%> milliseconds
^^Configuration 2: <% item.targetAlabel.1 %> with <% item.attributeBlabel.1 %>, <%
item.targetBlabel.1 %> with <% item.attributeAlabel.1 %>
^ <%block.incompatibletest.meanlatency%> milliseconds
^^Did you respond much more quickly on one of the configurations than the other? If so, that
configuration may be more consistent with your attitudes about these categories.
^^Thank you for your participation. Please press 'Continue' to end the test.
</page>
```

The "^" character is used to force a line break when the page is presented on screen. The page also includes several properties enclosed in "<% %>". When the page is displayed, these properties are replaced by the actual underlying property values. For example, <% block.compatibletest.meanlatency %> is replaced by the mean latency on the block named "compatibletest", and <% item.targetAlabel.1 %> is replaced by the first item in the item set named "targetAlabel".

Next we'll create the instruct element which determines how instruction pages are presented, and how the user navigates through them.

```
<instruct>
/ nextlabel = "Continue"
/ lastlabel = "Continue"
/ prevkey = (0)
/ inputdevice = mouse
/ windowsize = (90%, 90%)
/ screencolor = (0,0,0)
/ fontstyle = ("Arial", 3%)
/ txcolor = (255, 255, 255)
</instruct>
```

The `inputdevice` attribute specifies that users can navigate through the instructions by clicking the mouse. The `nextlabel` and `lastlabel` specifies the text label for the navigation button that advances to the next page, or that advances past the last page. By setting `prevkey` to "0", we ensure that users can not navigate backwards through the pages. Finally, we've defined look and feel of instruction pages using the `screencolor`, `fontstyle`, and `windowsize` commands.

3-Creating Trials

The IAT task requires that we create trial elements that present the stimuli representing the target and attribute categories and gather classification responses to those stimuli. There are six types of trials used in this task depending on which category of stimulus is presented and which response key is assigned as the correct classification of the category.

First, let's define trials involving good words:

```
<trial attributeA>  
/ validresponse = ("E", "I")  
/ correctresponse = ("E")  
/ stimulusframes = [1 = attributeA]  
/ posttrialpause = 250  
</trial>
```

The trial element's name is `attributeA`, which is also the name of the text element containing the good attribute words. The `validresponse` command indicates that participants may respond by pressing the "E" or the "I" key on the keyboard. The "E" key is considered a correct response as specified by the `correctresponse` command. The `posttrialpause` attribute specifies that after the response, Inquisit inserts a 250 ms pause before advancing to the next trial.

The definition of the other trial elements are the similar to `attributeA`, differing only in the type of stimulus presented and the response that's considered correct.

Here's the definition of trials with bad words where "I" is a correct response:

```
<trial attributeB>  
/ validresponse = ("E", "I")  
/ correctresponse = ("I")  
/ stimulusframes = [1 = attributeB]  
/ posttrialpause = 250  
</trial>
```

Next come trials with insect pictures classified with the "E" key:

```
<trial targetBleft>
/ validresponse = ("E", "I")
/ correctresponse = ("E")
/ stimulusframes = [1 = targetB]
/ posttrialpause = 250
</trial>
```

Trials with insect pictures classified with the "I" key:

```
<trial targetBright>
/ validresponse = ("E", "I")
/ correctresponse = ("I")
/ stimulusframes = [1 = targetB]
/ posttrialpause = 250
</trial>
```

Trials with flower pictures classified with the "E" key:

```
<trial targetAleft>
/ validresponse = ("E", "I")
/ correctresponse = ("E")
/ stimulusframes = [1 = targetA]
/ posttrialpause = 250
</trial>
```

Trials with flower pictures classified with the "I" key:

```
<trial targetAright>
/ validresponse = ("E", "I")
/ correctresponse = ("I")
/ stimulusframes = [1 = targetA]
/ posttrialpause = 250
</trial>
```

The trials above capture the different combinations of stimulus category and correct response in the IAT. Our script will define one additional trial used to present task instructions to participants. This trial is defined as follows:

```
<trial instructions>
/ stimulustimes = [1=instructions, spacebar]
/ correctresponse = (" ")
/ errormessage = false
/ recorddata = false
</trial>
```

The *stimulustimes* attribute specifies that the trial presents two text stimuli, one called "instructions" which contains the IAT task instructions, and another called "spacebar" which informs the participant they can press the spacebar to advance to the next trial. The *correctresponse* attribute indicates that pressing the spacebar key is the only correct response, and since the trial contains no *validresponse* definition, the spacebar is the only valid response as well. No error feedback is presented on this trial. Since the trial does not gather any data of interest, we've set *recorddata* to false so that the data for this trial (e.g., response, latency, stimuli, etc.) are not recorded to the data file. This helps us keep our data files concise and clean, and it saves us the trouble of having to filter out this data later.

4-Creating Blocks

Next we'll define the different kinds of blocks of trials used in the IAT. Blocks represent sequences of trials that can be in random or predetermined order. For this experiment, 11 block elements will be defined, 7 for practice at the IAT task, 2 for IAT data collection, and 2 for presenting task instructions.

First, let's define the practice block element for classification of the attribute categories, good and bad.

```
<block attributepractice>
/ bgstim = (attributeAleft, attributeBright)
/ trials = [1=instructions;2-21 = noreplace(attributeA, attributeB)]
/ errormessage = true(error,200)
/ responsemode = correct
/ recorddata = false
</block>
```

This block element is named "attributepractice". The *trials* attribute specifies that the block runs 1 instruction trial followed by 20 trials randomly selected without replacement from the two trial types "attributeA" and "attributeB", which present good and bad words respectively. The selection algorithm guarantees that both trial types will be run 10 times each. The *bgstim* attribute specifies that the "attributeA" and "attributeB" category labels are presented on the screen as background stimuli to remind participants how the response keys map to the categories. The *errormessage* attribute presents the stimulus named "error" (our big red X) for 200 ms whenever subjects respond incorrectly. The *responsemode* for the block is set to correct, which means participants must give the correct response to advance to the next trial, even if their initial response was incorrect. Finally, *recorddata* is set to false so that our data file isn't cluttered up with practice data from this block.

The rest of the blocks have a similar pattern. Next, let's define the practice blocks used for target categories. There are two such blocks for the two possible key assignments. Here is the practice block on which insects are classified with the right key and flowers with the left:

```
<block targetcompatiblepractice>
/ bgstim = (targetAleft, targetBright)
/ trials = [1=instructions;2-21 = noreplace(targetAleft, targetBright)]
/ errormessage = true(error,200)
/ responsemode = correct
/ recorddata = false
</block>
```

Now, let's define a practice block on which insects are classified with the left key and flowers with the right:

```
<block targetincompatiblepractice>
/ bgstim = (targetAright, targetBleft)
/ trials = [1=instructions;2-21 = noreplace(targetAright, targetBleft)]
/ errormessage = true(error,200)
/ responsemode = correct
/ recorddata = false
</block>
```

Next, let's define the practice blocks used after the key assignments for the target categories are switched. First, we'll define the block on which insects are classified with the left key and flowers with the right:

```
<block targetincompatiblepracticeswitch>
/ bgstim = (targetAleft, targetBright)
/ trials = [1=instructions;2-41 = noreplace(targetAleft, targetBright)]
/ errormessage = true(error,200)
/ responsemode = correct
/ recorddata = false
</block>
```

Next we'll define the opposite key assignments:

```
<block targetincompatiblepracticeswitch>
/ bgstim = (targetAright, targetBleft)
/ trials = [1=instructions;2-41 = noreplace(targetAright, targetBleft)]
/ errormessage = true(error,200)
/ responsemode = correct
/ recorddata = false
</block>
```


Notice that there are 40 practice trials in these blocks rather than 20. The extra trials are included to help subjects unlearn the key assignments from the previous blocks.

We've defined two blocks for the initial key assignment, and two more blocks for the key assignment after the switch. Since each subject can only have one initial assignment and one switched assignment, each subject will only encounter one of them depending on which key assignment condition they are assigned to.

The practice blocks above handle single category judgements, so we still need to define the practice for the mixed judgment blocks in which participants have to classify both attribute and target stimuli:

```
<block compatiblepractice>
/ bgstim = (targetAleft, orleft, attributeAleftmixed, targetBright, orright, attributeBrightmixed)
/ trials = [1=instructions;
  3,5,7,9,11,13,15,17,19,21= noreplace(targetAleft, targetBright);
  2,4,6,8,10,12,14,16,18,20 = noreplace(attributeA, attributeB)]
/ errormessage = true(error,200)
/ responsemode = correct
/ recorddata = false
</block>
```

```
<block incompatiblepractice>
/ bgstim = (targetBleft, orleft, attributeAleftmixed, targetAright, orright, attributeBrightmixed)
/ trials = [1=instructions;
  3,5,7,9,11,13,15,17,19,21 = noreplace(targetBleft, targetAright);
  2,4,6,8,10,12,14,16,18,20 = noreplace(attributeA, attributeB)]
/ errormessage = true(error,200)
/ responsemode = correct
/ recorddata = false
</block>
```

These are similar to previous practice blocks. One notable difference is that the *bgstim* attribute presents both target and attribute labels. Another difference is how the sequence of trials are defined. On odd numbered trials (excluding the instructions trial), the block runs a randomly selected target classification trial. On even numbered trials, a randomly selected attribute trial is run. This ensures that the participant does not encounter a run of trials in which they are making only flower/insect judgments or only bad/good judgments.

That does it for practice blocks. Lets define the test blocks. There are two such blocks, one using a "compatible" (i.e., stereotype consistent) pairing of target and attribute categories, and the other using the incompatible pairing:

```
<block compatibletest>
/ bgstim = (targetAleft, orleft, attributeAleftmixed, targetBright, orright, attributeBrightmixed)
/ trials = [
  2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40 = noreplace(targetAleft, targetBright);
  1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39 = noreplace(attributeA, attributeB)]
/ errormessage = true(error,200)
/ responsemode = correct
</block>
```

```
<block incompatibletest>
/ bgstim = (targetBleft, orleft, attributeAleftmixed, targetAright, orright, attributeBrightmixed)
/ trials = [
  2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40 = noreplace(targetBleft, targetAright);
  1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39 = noreplace(attributeA, attributeB)]
/ errormessage = true(error,200)
/ responsemode = correct
</block>
```

These blocks differ from the practice blocks in that they run 40 trials rather than 20, they do not have *recorddata* set to false, and they include no instruction trials. Instructions are instead displayed in a special instructions block. The reason for presenting the instruction trial in a separate block is because our summary page that we created earlier reports the average response latency score for the entire test block. If the test block included an instruction trial, the latency on this trial would also be included in the average. Since we want to report the average latency for test trials only and not instruction trials, we pulled the instruction trial out of the test block and put it into its own instruction block. The instruction blocks are defined below:

```
<block compatibletestinstructions>
/ bgstim = (targetAleft, orleft, attributeAleftmixed, targetBright, orright, attributeBrightmixed)
/ trials = [1=instructions]
/ recorddata = false
</block>
```

```
<block incompatibletestinstructions>
/ bgstim = (targetBleft, orleft, attributeAleftmixed, targetAright, orright, attributeBrightmixed)
/ trials = [1=instructions]
/ recorddata = false
</block>
```

That concludes our block definitions.

5-Creating an Expt

The `expt` element defines the sequence in which blocks are run. For our picture IAT, the `expt` element is defined as follows:

```
<expt>
/ blocks = [1=attributepractice; 2=block2; 3=block3; 4=block4; 5=block5; 6=block6; 7=block7;
8=block8; 9=block9]
/ postinstructions = (summary)
</expt>
```

The `expt` element is pretty simple. The `blocks` attribute specifies a sequence of 9 blocks. The first block is "attributepractice" block in which subjects practice classifying the good and bad word stimuli. Blocks 2 through 9 are set to between-subject variables named "block2", "block3", "block4", "block5", etc. When the script is run, these variables will be set to the names of real blocks depending on the subject number that is assigned. By using between-subject variables, the script counterbalances the order in which the test blocks are run across subjects so that half our subjects run the compatible pairing first, and the other half runs the incompatible pairing.

The between-subject variables are defined as follows:

```
<variables>
/ group = (1 of 2) (block2=targetcompatiblepractice; block3=compatiblepractice;
block4=compatibletestinstructions; block5=compatibletest; block6=targetincompatiblepractice;
block7=incompatiblepractice; block8=incompatibletestinstructions; block9=incompatibletest]
/ group = (2 of 2) (block2=targetincompatiblepractice; block3=incompatiblepractice;
block4=incompatibletestinstructions; block5=incompatibletest;
block6=targetcompatiblepractice; block7=compatiblepractice;
block8=compatibletestinstructions; block9=compatibletest]
</variables>
```

The `variables` element defines between-subject variables based on the subject number that was entered when the experiment is run. The first `group` attribute specifies the variable values for odd numbered subjects (i.e., the first of every two subjects). For odd-numbered subjects, `block2` is `targetcompatiblepractice`, `block3` is `compatiblepractice`, `block4` is `compatibletestinstructions`, and so on. Thus, odd numbered subjects perform classifications with the compatible pairing first. For even-numbered subjects, the incompatible pairing comes first.

By default, Inquisit will save a lot of data to the data file, much of which isn't relevant to the IAT. Although there's not much harm in having this data around, we can save ourselves some time and disk space by telling Inquisit to save just the data we care about. We do this in the `data` element as follows:

```
<data>
```

```
/ columns = (date time subject blockcode blocknum trialcode trialnum response correct latency  
stimulusnumber1 stimulusitem1 stimulusnumber2 stimulusitem2)
```

```
</data>
```

The *columns* attribute lists the data columns to save. All other data columns will not be saved.

Finally, we'll specify some default settings that apply to this script using the *defaults* element.

```
<defaults>
```

```
/ screencolor = (0,0,0)
```

```
/ txbgcolor = (0,0,0)
```

```
/ txcolor = (255, 255, 255)
```

```
/ fontstyle = ("Arial", 5%)
```

```
</defaults>
```

The *screencolor* attribute sets the color of the screen throughout the experiment to black. The *txcolor* and *txbgcolor* attributes specify the foreground and background colors for text stimuli as white text on a black background. The *fontstyle* attribute specifies that all text elements should be presented in Arial font at 5% of the screen height unless otherwise specified.

Our Picture IAT is now complete. You can run the experiment by selecting the "Run" command on the "Experiment" menu.

Tutorial: Simple Implicit Attitude Task

This tutorial builds a simplified Implicit Attitude Task (IAT). A number of standard IAT procedures have been omitted for the sake of illustrating basic Inquisit programming concepts. *The script produced by this tutorial is provided for instructional purposes only and should not be used for research.* To learn how to create a standard IAT, please see the [Standard IAT Tutorial](#) or the [Picture IAT Tutorial](#).

On the following pages, Inquisit commands are printed in blue, and comments are printed in black:

Steps

1. [Creating Text Stimuli](#)
2. [Creating Instructions](#)
3. [Creating Trials](#)
4. [Creating Blocks](#)
5. [Creating an Experiment](#)

1-Creating Text Stimuli

The first step in building an experiment is to define all of the stimuli. Stimuli include text or pictures to be shown on a given trial, background text that remains on the screen throughout a block of trials, or a feedback text shown to the subject to indicate when to respond and whether their response was correct or incorrect.

First, let's define the pleasant words:

```
<text pleasant>  
/ items = pleasant  
</text>
```

This text element defines a set of text stimuli named "pleasant" that has one attribute, *items*. The *items* attribute indicates where the text items are defined. In this case, they are defined in an item element named "pleasant" somewhere else in the script (more on this below). There are a number of other attributes that could be specified for our text stimulus, including attributes for controlling the color, background color, screen position, and font. However, we'll just use the defaults of black text on a white background presented in the middle of the screen.

Now, lets define the items for this text element:

```
<item pleasant>  
/ 1 = " HONOR  "  
/ 2 = " LUCKY  "  
/ 3 = " DIAMOND "  
/ 4 = " LOYAL  "  
/ 5 = " FREEDOM "  
/ 6 = " RAINBOW "  
/ 7 = " LOVE   "  
/ 8 = " HONEST "  
/ 9 = " PEACE  "  
/10 = " HEAVEN "  
</item>
```

This item element is named "pleasant", which matches the name specified in the items attribute of the text element above. The item set consists of ten pleasant words. Note that the words are padded with spaces so that they are all of equal length when presented in a fixed width font.

Now, lets define the rest of the stimulus categories. First, we'll define the unpleasant words:

```
<text unpleasant>  
/ items = unpleasant  
</text>  
and the unpleasant items.
```

```
<item unpleasant>  
/ 1 = " EVIL   "  
/ 2 = " CANCER "  
/ 3 = " SICKNESS "  
/ 4 = " DISASTER "  
/ 5 = " POVERTY "  
/ 6 = " VOMIT   "  
/ 7 = " BOMB    "  
/ 8 = " ROTTEN  "  
/ 9 = " ABUSE   "  
/10 = " MURDER  "  
</item>
```

Next, we'll define the flowers:

```
<text flower>
/ items = flowers
</text>
and the flower items.
<item flowers>
/ 1 = "   ROSE   "
/ 2 = "  BEGONIA  "
/ 3 = "  VIOLET   "
/ 4 = "   DAISY   "
/ 5 = " GERANIUM  "
/ 6 = "   TULIP   "
/ 7 = " CARNATION "
/ 8 = " DAFFODIL  "
/ 9 = "   LILAC   "
/ 10= "  PANSY   "
</item>
```

Finally, we'll define the insects:

```
<text insect>
/ items = insects
</text>
and insect items.
<item insects>
/ 1 = "   ANT    "
/ 2 = "  LOCUST  "
/ 3 = "   BEE    "
/ 4 = "  HORNET  "
/ 5 = "   WASP   "
/ 6 = "  SPIDER  "
/ 7 = " CENTIPEDE "
/ 8 = " COCKROACH "
/ 9 = "  BEDBUG  "
/ 10= " LADYBUG  "
</item>
```

When creating an IAT, it's a good idea to include instruction text that reminds participants how to respond to the various stimulus categories. We can do this by presenting text on the screen that are shown in the background throughout a block of trials. So, let's create the instruction text stimuli that remind the subject to press the "a" key for unpleasant and the "5" key for pleasant.

```
<text pleasantreminder>
/ items = ("Press 'a' for pleasant")
/ position = (75, 25)
/ txcolor = (0, 0, 255)
</text>
```

The reminder stimulus is a bit different than the previous stimuli. First, rather than defining the items in a separate element, we've simply listed the single item directly in the attribute. This inline syntax is a convenient way to define small item sets for things like instructions, focus stimuli, and masks. Also, the *position* attribute specifies that the text should be displayed on the upper right of the screen rather than in the default center position. Specifically, the stimulus is positioned 75% of way across the screen (from left to right), and 25% percent of the way down the screen (from top to bottom). Finally, the *txcolor* attribute specifies that the text should be blue rather than the default color black. Colors in Inquisit are specified as a mix of red, green, and blue components; the *txcolor* attribute specifies 0 intensity for red and green components, and the maximum intensity 255 for the blue component, producing a nice blue color.

Now, lets define the unpleasant reminder, which will be displayed in the upper left quadrant of the screen.

```
<text unpleasantreminder>
/ items = ("Press '5' for unpleasant")
/ position = (25, 25)
/ txcolor = (0, 0, 255)
</text>
```

and the rest of the reminders:

```
<text flowerleft>
/ items = ("Press 'a' for flowers")
/ position = (25, 25)
/ txcolor = (0, 0, 255)
</text>
<text flowerright>
/ items = ("Press '5' for flowers")
/ position = (75, 25)
/ txcolor = (0, 0, 255)
</text>
<text insectleft>
/ items = ("Press 'a' for insects")
/ position = (25, 25)
/ txcolor = (0, 0, 255)
</text>
```



```
<text insectright>  
/ items = ("Press '5' for insects")  
/ position = (75, 25)  
/ txcolor = (0, 0, 255)  
</text>  
<text pleasant_flower>  
/ items = ("Press '5' for pleasant or flowers")  
/ position = (75, 25)  
/ txcolor = (0, 0, 255)  
</text>  
<text pleasant_insect>  
/ items = ("Press '5' for pleasant or insects")  
/ position = (75, 25)  
/ txcolor = (0, 0, 255)  
</text>  
<text unpleasant_flower>  
/ items = ("Press 'a' for unpleasant or flower")  
/ position = (25, 25)  
/ txcolor = (0, 0, 255)  
</text>  
<text unpleasant_insect>  
/ items = ("Press 'a' for unpleasant or insect")  
/ position = (25, 25)  
/ txcolor = (0, 0, 255)  
</text>
```

Finally, lets define an error message stimulus to show subjects whenever they incorrectly classify a target stimulus:

```
<text errormessage>  
/ items = ("  ERROR  ")  
/ txcolor = (255, 0, 0)  
</text>
```

The "errormessage" text element uses the *txcolor* attribute to set the red component to 255 and the green and blue components to 0, producing a rich red color.

2-Creating Instructions

Now lets define a set of instruction pages that inform the subject how to perform the task. Defining the instruction pages is easy using the page element. First, we'll define a simple welcome page.

```
<page intro>
^^^^^^^ Implicit Association Test
^^Welcome and thank you for participating.
</page>
```

Note that the "^" character is used to force a line break. Otherwise, lines of text are word-wrapped. Now we'll define the rest of the instruction pages:

```
<page up>
The tasks that you will be doing in this experiment involve CATEGORY JUDGMENT. On each trial,
a stimulus will be displayed, and you must assign it to one of two categories. You should respond
AS RAPIDLY AS POSSIBLE in categorizing each stimulus, but don't respond so fast that you make
many errors. (Occasional errors are okay.)^^
The two categories that you are to distinguish are:^^
UNPLEASANT vs. PLEASANT words.^^
Press the "a' key if the stimulus is an UNPLEASANT word.^^
But press "5' key if the stimulus is a PLEASANT word.^^
</page>
```

```
<page if>
The two categories that you are to distinguish are:^^
INSECTS vs. FLOWERS.^^
Press the "a' key if the stimulus is an INSECT.^^
But press "5' key if the stimulus is a FLOWER.^^
</page>
```

```
<page fi>
The two categories that you are to distinguish are:^^
FLOWERS vs. INSECTS.^^
Press the "a' key if the stimulus is a FLOWER.^^
But press "5' key if the stimulus is an INSECT.^^
</page>
```

<page compatible>

The four categories that you are to distinguish are:^^

UNPLEASANT vs. PLEASANT words^

or^

INSECTS vs. FLOWERS.^^

Press the "a" key if the stimulus is^

an UNPLEASANT word or an INSECT.^^

But press "5" key if the stimulus is^

a PLEASANT word or a FLOWER.^^

</page>

<page incompatible>

The four categories that you are to distinguish are:^^

UNPLEASANT vs. PLEASANT words^

or^

FLOWERS vs. INSECTS.^^

Press the "a" key if the stimulus is^

an UNPLEASANT word or a FLOWER.^^

But press "5" key if the stimulus is^

a PLEASANT word or an INSECT.^^

</page>

<page end>

The Implicit Association Test is now concluded.

If you have any questions or reactions to the experiment, please discuss them with the experimenter.

</page>

Finally, we'll specify how participants can navigate through the instruction pages using the instruct element. A script should have only one such element.

<instruct>

/ nextkey = ("5")

/ prevkey = ("a")

</instruct>

The *nextkey* attribute indicates that participants must press the "5" key to advance to the next page, and the *prevkey* attribute specifies pressing the "a" key goes back to the previous key.

3-Creating Trials

The next step is to define the different kinds of trials that will be used in the IAT task. Trial elements control which stimuli are presented and how the subject may respond to those stimuli. There are six types of trials used in this task depending on which semantic category of stimulus is presented and which response key is assigned as the correct classification of the category.

First, let's define trials involving pleasant words, which are always assigned to the right response key.

```
<trial pleasant>
/ stimulusframes = [1=pleasant]
/ validresponse = ("a", "5")
/ correctresponse = ("5")
</trial>
```

The trial element's name is *pleasant*. On each line of data in the data file corresponding to this type of trial, this trial name is written.

The *stimulusframes* attribute defines the stimulus presentation sequence of the trial. The entire presentation sequence will consist of as many frames as are specified in the frames attribute (only 1 in this case). A pleasant word is presented on the first frame, after which Inquisit begins waiting for (and timing) the subject's response.

The *validresponse* attribute indicates that the subject may respond by pressing either the "a" or the "5" key, after which Inquisit will advance to the next trial. The *correctresponse* attribute indicates that only the "5" key is considered a correct response on this type of trial.

The definition of the other trial elements are the similar to *pleasant*, differing only in the type of stimulus presented and the response that's considered correct.

Here's the definition of trials with unpleasant words where "a" is a correct response:

```
<trial unpleasant>
/ validresponse = ("a", "5")
/ correctresponse = ("a")
/ stimulusframes = [1=unpleasant]
</trial>
```

Next come trials with insect names classified with the "5" key:

```
<trial insright>
/ validresponse = ("a", "5")
/ correctresponse = ("5")
/ stimulusframes = [1=insect]
</trial>
```

Trials with insect names classified with the "a" key:

```
<trial insleft>
/ validresponse = ("a", "5")
/ correctresponse = ("a")
/ stimulusframes = [1=insect]
</trial>
```

Trials with flower names classified with the "5" key:

```
<trial flowright>
/ validresponse = ("a", "5")
/ correctresponse = ("5")
/ stimulusframes = [1=flower]
</trial>
```

Trials with flower names classified with the "a" key:

```
<trial flowleft>
/ validresponse = ("a", "5")
/ correctresponse = ("a")
/ stimulusframes = [1=flower]
</trial>
```

The trials above capture the different combinations of stimulus category and correct response in the IAT.

4-Creating Blocks

The next step is to define the different kinds of blocks that will be used in the experiment. Blocks represent sequences of trials that can be in random or fixed order. For this experiment, 5 block elements will be defined, 3 for practice trials and 2 for data collection.

First, let's define the practice block element for classification of pleasant and unpleasant words.

```
<block up_practice>
/ trials = [1-20 = noreplace(pleasant, unpleasant)]
/ bgstim = (pleasantreminder, unpleasantreminder)
/ preinstructions = (up)
/ errormessage = (errormessage, 200)
/ blockfeedback = (latency, correct)
</block>
```

This block element is named "up_practice". The *trials* attribute specifies that the block runs 20 trials randomly selected without replacement from the two trial types "pleasant" and "unpleasant". The selection algorithm guarantees that both trial types will be run an equal number of times (10 time each). The *bgstim* attribute specifies that the "pleasantreminder" and "unpleasantreminder" instruction text stimuli are presented on the screen as background stimuli. The *preinstructions* attribute displays 3 pages of instructions ("intro1", "intro2", and "intro3") before running the trials. The *errormessage* attribute presents the "errormessage" stimulus for 200 ms whenever subjects respond incorrectly. Finally, the *blockfeedback* attribute specifies that after the block is over, subjects will be shown their mean latency and percent correct for the block.

The rest of the blocks have a similar pattern. Next, lets define a practice block on which insects are classified with the left key and flowers with the right:

```
<block if_practice>
/ trials = [1-20 = noreplace(insleft, flowright)]
/ bgstim = (insectleft, flowerright)
/ preinstructions = (if)
/ errormessage = (errormessage, 200)
/ blockfeedback = (latency, correct)
</block>
```

Now, lets define a practice block on which insects are classified with the right key and flowers with the left:

```
<block fi_practice>
/ trials = [1-20 = noreplace(insright, flowleft)]
/ bgstim = (insectright, flowerleft)
/ preinstructions = (fi)
/ errormessage = (errormessage, 200)
/ blockfeedback = (latency, correct)
</block>
```

Next, lets define the "compatible" test block. Note that on test blocks we no longer display an error message for incorrect responses:

```
<block compatible>
/ trials = [1-40 = noreplace(insleft, flowright, pleasant, unpleasant)]
/ bgstim = (unpleasant_insect, pleasant_flower)
/ preinstructions = (compatible)
/ blockfeedback = (latency, correct)
</block>
```

Finally, lets define the "incompatible" block:

```
<block incompatible>
/ trials = [1-40 = noreplace(insright, flowleft, pleasant, unpleasant)]
/ bgstim = (pleasant_insect, unpleasant_flower)
/ preinstructions = (incompatible)
/ blockfeedback = (latency, correct)
</block>
```

5-Creating an Expt

The next step is to define an expt element that defines the flow of blocks in the experiment. The expt element is defined as follows:

```
<expt>
/ preinstructions = (intro)
/ postinstructions = (end)
/ blocks = [1=up_practice; 2=block2; 3=block3; 4=block4; 5=block5]
</expt>
```

The expt element is simple. The *preinstructions* attribute begins the expt by showing subjects a page of instructions, "intro". The *postinstructions* attribute specifies final instruction page named "end" to be displayed at the conclusion of the experiment. The *blocks* attribute specifies a total of 5 blocks. The first block is the "up_practice" block in which subjects practice classifying the pleasant and unpleasant stimuli. Blocks 2 through 5 are set to between-subject variables named "block2", "block3", "block4", and "block5", all of which are defined below. These between-subject variables allow the experiment to counterbalance the order in which the test blocks are run across subjects.

Next, we'll define the between-subject variables used above:

```
<variables>  
/ group = (1 of 2) (block2=fi_practice, block3=incompatible, block4=if_practice,  
block5=compatible)  
/ group = (2 of 2) (block2=if_practice, block3=compatible, block4=fi_practice,  
block5=incompatible)  
</variables>
```

The *variables* element defines between-subject variables based on the subject number that was entered when the experiment is run. The first *group* attribute specifies the variable values for odd numbered subjects. For odd-numbered subjects, *block2* is *fi_practice*, *block3* is *incompatible*, *block4* is *if_practice*, and *block5* is *compatible*. For even-numbered subjects, *block2* is *if_practice*, *block3* is *compatible*, *block4* is *fi_practice*, and *block5* is *incompatible*.

With that, the script is essentially complete. However, we'll do a little fine tuning by specifying some default settings using the *defaults* element.

```
<defaults>  
/ screencolor = (175, 175, 255)  
/ fontstyle = ("Courier New", 14pt)  
</defaults>
```

The *screencolor* attribute sets the color of the screen throughout the experiment to light blue. The *fontstyle* attribute specifies that all stimulus and instruction text should be displayed in a 14pt Courier New font. You can specify the font attribute using Inquisit's Font Wizard, available from the Tools menu. The wizard allows you to pick a font using the standard font dialog, and will spit the corresponding attribute definition into your script.

The experiment is now complete! You can run the experiment by selecting the "Run" command on the "Experiment" menu.

Tutorial: Subliminal Priming Task

This tutorial explains how to build a simple subliminal priming experiment such as that described by Draine and Greenwald (Journal of Experimental Psychology: General, 1998). On each trial, subjects are shown a subliminal (masked) prime word followed immediately by a target word. Primes and targets are divided into two definitional categories: pleasant (e.g., love) or unpleasant (e.g., death). Subjects are instructed to ignore the primes and classify the targets as pleasant (by pressing the "5" key on the number pad) or unpleasant (by pressing the "a" key). Each trial falls into one of four experimental conditions depending on the category (pleasant or unpleasant) of prime and target presented:

- pleasant prime with pleasant target (congruent)
- pleasant prime with unpleasant target (incongruent)
- unpleasant prime with pleasant target (incongruent)
- unpleasant prime with unpleasant target (congruent)

Longer Reaction Times and/or higher error rates on trials in which the prime and target are *incongruent* compared to the condition in which the prime and target are *congruent*, suggest the presence of subliminal priming.

Steps

1. Creating Text Stimuli
2. Creating Instructions
3. Creating Trials
4. Creating Blocks
5. Creating an Experiment

1-Creating Text Stimuli

The first step in building an experiment is to create the experimental stimuli and specify how they should be presented. Stimuli will typically consist of text or pictures presented during a trial, instruction text that remains on the screen throughout a block of trials, and feedback messages that indicate when to respond and whether or not a response was correct.

First, let's create a text element that defines the pleasant prime words:

```
<text pleasantprime>
/ items = pleasant
</text>
```

The text element's name is "pleasantprime". The element could be named anything we wish, but it's a good idea to pick a simple, descriptive name. It has a single attribute called "items" that specifies where the actual items are located. In this case, we have specified that the items are located in an item element called "pleasant" that we will create a bit later.

The text element allows us to define other attributes including color and screen location. In this element, we will use the default values. (The default color is black, and the default position is the center of the screen.) Later, we'll demonstrate how to change these values.

In the example above, we defined pleasant prime stimuli. Next, we'll define pleasant target stimuli:

```
<text pleasanttarget>
/ items = pleasant
</text>
```

With the exception of its name, this text element is identical to the pleasantprime element. Note that the *items* attribute is set to the same item list (that we'll create in a moment) named "pleasant". This element will use those same items as targets.

Now, let's define the items used by both the pleasantprime and pleasanttarget text elements:

```
<item pleasant>
/ 1 = " HONOR  "
/ 2 = " LUCKY  "
/ 3 = " DIAMOND "
/ 4 = " LOYAL  "
/ 5 = " FREEDOM "
/ 6 = " RAINBOW "
/ 7 = " LOVE   "
```

```
/ 8 = "   HONEST   "
/ 9 = "   PEACE   "
/10 = "  HEAVEN   "
</item>
```

Pretty simple isn't it? Notice the opening and closing lines: <item pleasant> and </item> The name of this element is pleasant. Notice that there are 8 spaces on either side of the items (words). This is done to center the words in the presentation box during a given trial.

Now, lets define the unpleasant primes and targets:

```
<text unpleasantprime>
/ items = unpleasant
</text>
<text unpleasanttarget>
/ items = unpleasant
</text>
```

These text elements similar to the previous ones, except that they use "unpleasant" items. Let's create the unpleasant items:

```
<item unpleasant>
/ 1 = "   EVIL   "
/ 2 = "  CANCER  "
/ 3 = " SICKNESS "
/ 4 = " DISASTER "
/ 5 = " POVERTY  "
/ 6 = "  VOMIT   "
/ 7 = "   BOMB   "
/ 8 = "  ROTTEN  "
/ 9 = "  ABUSE   "
/10 = " MURDER   "
</item>
```

Let's review what we've covered so far.

1. Inquisit uses a set of instructions to control the flow of an experiment called a script.
2. A script consists of elements.
3. Each element has its own name by which other elements can refer to it.
4. Elements have attributes that control specific properties of the element (e.g. font size and color).
5. Some attributes of an element can refer to other elements in the script by name.

We aren't done with the stimuli yet. We still need to define the forward and backward, the error messages, and create background messages. Here is the text element that will serve as the forward masks of the primes, called "forwardmask":

```
<text forwardmask>
/ items = (" KQHYTPDQFPBYL ", " PYLDQFBYDQKPH ")
</text>
```

This text element consists of two items, "KQHYTPDQFPBYL" and "PYLDQFBYDQKPH". Note that the items in this case are defined directly inside the forwardmask text element. You can define items this way or by using an items element as was done previously. A good rule of thumb is to use the items element for large item sets, or for item sets that will be shared by multiple text elements (for example, the pleasant items were used by both pleasantprime and pleasanttarget). For small item sets such as the two forward masks, or for items sets that are only used by one text element, it is usually more convenient to define such items directly inline.

Here is the text element that will define the stimuli that will serve as backward masks of the primes. Let's call it "backwardmask". The backward mask is very similar to the forward mask:

```
<text backwardmask>
/ items = (" PYLDQFBYDQKPH ", " KQHYTPDQFPBYL ")
/ select = current (forwardmask)
</text>
```

Note the addition of the *select* attribute set to the *current* option. By default, the select attribute is set to *noreplace* and items are selected without replacement on each trial. The *current* setting links the selection of the backwardmask item on each to that of the forwardmask item. Thus, for trials on which both a forward and backward mask are presented, if the first forward mask item is selected and presented on that trial, then the first backward mask item will also be presented. If the second forward mask was selected and presented, then the second backward mask will also be presented. Thus, each forward mask item has a complimentary backward mask item that always appears in conjunction with it. By linking these two stimuli, the forward and backward masks on a given trial will never be identical since the order of the two items is reversed.

Now, lets create stimuli (text) to be shown in the background throughout an entire block of trials. These stimuli will serve as reminders to the subject to press the "a" key for unpleasant and the "5" key for pleasant.

```
<text pleasantreminder>
/ items = ("5 = pleasant")
/ position = (75, 25)
</text>
```

This stimulus is similar to the previous stimuli, except the *position* attribute is no longer set to the default (center). Inquisit specifies screen position using a coordinates system ranging from 0 to 100 on both the horizontal and vertical axes. The upper left corner of the screen is (0, 0), and the lower right corner of the screen is (100, 100). The center of the screen is (50, 50). So, the coordinates of (75, 25) used above in pleasantreminder will place the stimuli above and to the right of the center of the screen.

Now, let's define the unpleasant reminder, which will be displayed on the upper left region of the screen.

```
<text unpleasantreminder>
/ items = ("a = unpleasant")
/ position = (25, 25)
</text>
```

Finally, let's define a stimulus to show subjects whenever they incorrectly classify a target stimulus:

```
<text errormessage>
/ items = (" ERROR ")
/ color = (255, 0, 0)
</text>
```

The errormessage text element uses the *color* attribute. The color attribute takes three integers between 0 and 255 that define the intensity of the red, green, blue components of the color respectively. The red component is the maximum intensity, 255, whereas the green and blue components are 0. This combination produces a rich red color.

2-Creating Instructions

Next, let's define the instruction pages. First, we'll create an introduction page:

```
<page intro1>
The tasks that you will be doing in this experiment involve CATEGORY JUDGMENT.
On each trial, a stimulus will be displayed, and you must assign it to one of
two categories. You should respond AS RAPIDLY AS POSSIBLE in categorizing each stimulus,
but don't respond so fast that you make many errors. (Occasional errors are okay.)
^^
The two categories that you are to distinguish are:
^^
UNPLEASANT vs. PLEASANT words.
^^
Press the "a" key if the stimulus is an UNPLEASANT word.
^^
But press "5" key if the stimulus is a PLEASANT word.
^^
</page>
```

The page element doesn't have any attributes, but simply contains the content of the page. The special character "^" will force a line break when the page is displayed on the screen. Otherwise, the instructions are word wrapped inside the page area.

Now let's define the rest of the pages in the script:

```
<page intro2>
```

Just before each word that you are to categorize you will see one or more words and letter strings briefly flashed.^ ^

It is your task to IGNORE these briefly flashed stimuli. Respond only to the last, clearly visible word shown on each trial.

```
</page>
```

```
<page intro3>
```

When you press the "5" key, you will see a stimulus to which you should respond.^ ^

As a reminder of the instructions for responding:^ ^

Press the "a" key if the stimulus is an UNPLEASANT word.^ ^

Press "5" key if the stimulus is a PLEASANT word.^ ^

```
</page>
```

```
<page ready>
```

When you press the "5" key, a new block of trials at the same task as the last block will start.^ ^

Be ready for the first stimulus when you press the key.

```
</page>
```

```
<page end>
```

The experiment is now concluded. If you have any questions or reactions to the experiment, please discuss them with the experimenter.

```
</page>
```

Finally, we'll define an instruct element that specifies how subjects can navigate from page to page. A script should have only one such element.

```
<instruct>
```

```
/ nextkey = ("5")
```

```
/ prevkey = ("a")
```

```
</instruct>
```

The *nextkey* attribute specifies that subjects can press the "5" key to advance to the next instruction page, and "a" key to go back to a previous page.

3-Creating Trials

The next step is to define the different kinds of trials that will be used in the experiment. The types of trials you define will correspond to the different conditions of the experiment. This experiment has four conditions, one condition for each of the four possible combinations of prime and target categories.

First, let's define trials involving pleasant primes and pleasant targets by creating the following trial element.

```
<trial pp>
/ pretrialpause = 300
/ validresponse = ("a", "5")
/ correctresponse = ("5")
/ stimulusframes = [1=forwardmask; 10=pleasantprime; 13=backwardmask; 14=pleasanttarget]
/ posttrialpause = 100
</trial>
```

The trial element is called "pp", which is short for "pleasant pleasant" because the trial presents both a pleasant prime and pleasant target.

The *posttrialpause* attribute tells Inquisit to pause 300 milliseconds before each trial is executed.

The *validresponse* attribute specifies which keys a subject can press to register their response. Remember we already talked about these responses when the instructions were created. When the participant responds by pressing the "a" or the "5" key, Inquisit will advance to the next trial.

The *correctresponse* attribute tells Inquisit which responses are considered correct. In this trial, "5" is correct.

the *stimulusframes* attribute is slightly more technical because it is closely connected to how CRT and LCD monitors operate. Computer monitors repaint the screen from top to bottom according to a fixed interval called a "frame" (a.k.a., vertical retrace interval). Most standard monitors repaint the screen about every 10 to 17 milliseconds. To determine the frequency at which your CRT monitor repaints the screen, select the "Check Hardware" command from Inquisit's Tools menu. Inquisit will run at any frequency. If you decide you would like to change the retrace frequency of your video system, you should check the manufacturer's documentation for the specifics on your monitor. Typically, the frame rate can be controlled from Display settings within Windows.

Thus, the stimulus presentation sequence is defined in terms of discrete frames rather than times. The entire presentation sequence consists of as many frames as are specified in the frames attribute (14 in this case).

So, a forwardmask is presented at the onset of the first frame of the trial. This forwardmask remains on the screen until it is overwritten by a prime stimulus on the 10th frame. The prime stimulus remains on the screen for 3 frames (50 ms on a 60 hz monitor), before it is overwritten by a backward mask on the 13th frame. Finally, a target is presented on the 14th frame and remains on the screen until the subject responds.

So you see, you have to do a little calculating here to decide the time each frame is to be presented. 3 frames on a 60 hz monitor is the equivalent of 50 ms because:
3 (frames) x 16.7 (ms) = 50.1 (ms).

Finall, the *posttrialpause* attribute tells Inquisit to wait 100 ms after this trial before advancing to the next trial.

Remember, this was just one type of trial. We need to define 3 more trial types. The rest of the trials will be very similar to the first. Many experiments change only minor details from trial to trial; a word list, an order of presentation, response variables, etc. Once you have the first trial coded, you are ready to make the rest. It's very easy to do that in Inquisit. You can copy and paste your code for trial one and then just change the minor variables. Let's see what the other trial codes look like.

```
<trial pu>
/ pretrialpause = 300
/ validresponse = ("a", "5")
/ correctresponse = ("a")
/ stimulusframes = [1=forwardmask; 10=pleasantprime; 13=backwardmask;
14=unpleasanttarget]
/ posttrialpause = 100
</trial>
```

```
<trial up>
/ pretrialpause = 300
/ validresponse = ("a", "5")
/ correctresponse = ("5")
/ stimulusframes = [1=forwardmask; 10=unpleasantprime; 13=backwardmask;
14=pleasanttarget]
/ posttrialpause = 100
</trial>
```

```
<trial uu>
/ pretrialpause = 300
/ validresponse = ("a", "5")
```



```

/ correctresponse = ("a")
/ stimulusframes = [1=forwardmask; 10=unpleasantprime; 13=backwardmask;
14=unpleasanttarget]
/ posttrialpause = 100
</trial>

```

The three trials above differ from the original only by which stimuli they present and which response is considered correct. Together, the four trials capture the four combinations of pleasant and unpleasant prime and target stimuli.

4-Creating Blocks

The next step is to define the different kinds of blocks that will be used in the experiment. For this experiment, two block elements will be defined, one for practice trials and the other for data collection.

First, let's define the practice block element.

```

<block practice>
/ bgstim = (pleasantreminder, unpleasantreminder)
/ preinstructions = (intro1, intro2, intro3)
/ trials = [1-40 = noreplace(pp, pu, up, uu)]
/ errormessage = (errormessage, 200)
/ blockfeedback = (latency, correct)
</block>

```

The block is called "practice".

The *bgstim* attribute tells Inquisit to keep the pleasantreminder and unpleasantreminder on the screen during the block. Remember these stimulus elements were defined previously, we wanted to keep "a=unpleasant" and "5=pleasant" on the screen during the trials in the upper left and right quadrants respectively.

The *preinstructions* attribute refers lists some of the instructions pages created in the previous section. This command tells Inquisit to display the pages named intro1, intro2, and intro3 at the beginning of the block.

The *trials* attribute defines trials (40) for this practice block that are randomly without replacement selected from the four trial types: pp, pu, up, uu. This guarantees that each trial type will be presented 10 times.

The *errormessage* attribute indicates that when the participant responds incorrectly in this practice block, the errormessage text stimulus (previously defined) will be displayed for 200 ms.

Giving feedback during a practice trial is a good idea because you generally want the participant to learn the right way to perform their task. As you'll see, we remove this feedback in the data collection blocks.

The *blockfeedback* attribute specifies that after the block is over, subjects will be shown their mean latency and percent correct.

Now, let's define a data collection block.

```
<block data>
/ screencolor = (175, 175, 255)
/ bgstim = (pleasantreminder, unpleasantreminder)
/ preinstructions = (ready)
/ trials = [1-40 = random(pp, pu, up, uu)]
/ blockfeedback = (latency, correct)
</block>
```

This element is just like the practice element except that the participant is no longer given the feedback because we removed the *errormessage* attribute. Also, the block begins with a different instruction page called "ready".

5-Creating an Expt

Next, we'll need to define an *expt* element that specifies which blocks to run.

```
<expt>
/ blocks = [1 = practice; 2-5 = data]
/ postinstructions = (end)
</expt>
```

The *expt* element runs a total of 5 blocks. The first block is "practice", and the next four blocks are "data". After all the blocks have been run, a single page of instructions called *end* is displayed.

Finally, we'll want to set the default font and screen color for the experiment using the *defaults* element:

```
<defaults>
/ fontstyle = ("Courier New", 14pt)
/ screencolor = (150, 150, 150)
</defaults>
```

The *fontstyle* attribute sets the default font for all text stimuli and instruction pages to 14pt Courier New. You can use Inquisit's Font Wizard to generate the font selection for you.

To use the Font Wizard, place your cursor at the location in the script where you wish the font attribute to appear, then select the Font Wizard command from the Tools menu. A standard font dialog will appear, allowing you to select the font of your choice. The wizard will then inject the corresponding attribute definition into your script.

The *screencolor* attribute works just like the *txcolor* attribute we configured earlier. In this case, the red, green, and blue components are all set to 150, making the screen grey.

The experiment is complete!

Tutorial: Covert Attention Task

This tutorial builds a covert attention task. The task measures the effect of an unattended cue on spatial position judgments. Subjects perform the task by indicating whether a critical stimulus is presented on the left or right side of the screen by pressing the 'a' or 's' key on the keyboard respectively. Subjects are instructed to fixate their gaze on the center of the screen while performing the task, where an arrow is presented on each trial. On 80% of the trials, the arrow points in the direction where the critical stimulus is presented (compatible). On the remaining 20% of the trials, the arrow points in the opposite direction (incompatible). Covert attention to the arrow is measured by comparing the average response latency of the compatible and incompatible trials. Shorter mean latencies on compatible as compared to incompatible trials indicates that subjects are influenced by the unattended arrow stimulus.

Steps

1. [Creating Text Stimuli](#)
2. [Creating Instructions](#)
3. [Creating Trials](#)
4. [Creating Blocks](#)
5. [Creating an Experiment](#)

1-Creating Text and Picture Stimuli

The first step in building an experiment is to define all of the stimuli. Stimuli include text or pictures to be shown on a given trial, background text that remains on the screen throughout a block of trials, or a feedback text shown to the subject to indicate when to respond and whether their response was correct or incorrect.

Inquisit allows you to specify global default settings for stimuli and other parts of the experiment using the `<defaults>` element. For this script, we will set the default font for all text stimuli that we present, and we'll also set the background color for the screen to black.

```
<defaults>  
/ screencolor = (0, 0, 0)  
/ fontstyle = ("Arial", 20pt)  
</defaults>
```

Now, let's create the arrow stimuli to be presented at the fixation point in the center of the screen:

```
<picture leftarrow>  
/ items = ("http://www.millisecond.com/samples/covertattention/leftarrow.jpg")  
</picture>
```

```
<picture rightarrow>  
/ items = ("http://www.millisecond.com/samples/covertattention/rightarrow.jpg")  
</picture>
```

That was pretty easy. We simply created two picture stimuli called "leftarrow" and "rightarrow". Both stimuli consist of a single item picture item contained in the picture files "rightarrow.jpg" and "leftarrow.jpg" respectively. In this case, the files are downloaded from <http://www.millisecond.com/samples/covertattention>, but typically you would keep the files in the same folder as the script. By default, Inquisit presents the pictures in the center of the screen.

Now, let's define a text stimulus to serve as the fixation point itself.

```
<text fixation>  
/ items = ("+")  
/ color = (255, 255, 255)  
/ txbgcolor = (0,0,0)  
/ fontstyle = ("Arial", 30pt)  
/ erase = false  
</text>
```

The text is called "fixation" and it consists of a single item, "+". The foreground color is white, with the red, green, and blue values set to the maximum value of 255. You can use Inquisit's Color Wizard available on the Tools menu to get the red, green, and blue values for any color. The background color is black, with red, green, and blue values set to the minimum value of 0. The font is Arial. You can select a font using Inquisit's Font Wizard, also available on the Tools menu. Finally, the erase command indicates that the fixation point should not be erased.

Now, let's define text stimuli to be presented just below the fixation point that will serve as instruction reminders.

```
<text instructleft>  
/ items = ("Press A if the brightened box is on the left.")  
/ position = (50, 60)  
/ color = (255, 255, 255)  
/ txbgcolor = (0,0,0)  
</text>
```

```
<text instructright>
/ items = ("Press S if the brightened box is on the right.")
/ position = (50, 65)
/ color = (255, 255, 255)
/ txbgcolor = (0,0,0)
</text>
```

The instruction stimuli use the position command to present the stimuli just below the fixation point. Position is specified in terms of x and y coordinates. The unit of measurement is percentage, with 0 = top/left, 50 = center, and 100 = bottom right. The instruction reminders are presented horizontally centered and vertically 10 percentage points below center.

Now, let's define the target pictures. We will create four target stimuli, each of which presents a picture of a light yellow rectangle in one of the four corners of the screen.

```
<picture toplefttarget>
/ items = ("http://www.millisecond.com/samples/covertattention/targetrectangle.jpg")
/ position = (0, 0)
/ valign = top
/ halign = left
</picture>
```

```
<picture bottomlefttarget>
/ items = ("http://www.millisecond.com/samples/covertattention/targetrectangle.jpg")
/ position = (0, 100)
/ valign = bottom
/ halign = left
</picture>
```

```
<picture toprighttarget>
/ items = ("http://www.millisecond.com/samples/covertattention/targetrectangle.jpg")
/ position = (100, 0)
/ valign = top
/ halign = right
</picture>
```

```
<picture bottomrighttarget>
/ items = ("http://www.millisecond.com/samples/covertattention/targetrectangle.jpg")
/ position = (100, 100)
/ valign = bottom
/ halign = right
</picture>
```

Finally, we'll create the distractor stimuli. There are four distractors, each of which presents a dark yellow rectangle in the four corners of the screen.

```
<picture topleft>  
/ items = ("http://www.millisecond.com/samples/covertattention/rectangle.jpg")  
/ position = (0, 0)  
/ valign = top  
/ halign = left  
</picture>
```

```
<picture bottomleft>  
/ items = ("http://www.millisecond.com/samples/covertattention/rectangle.jpg")  
/ position = (0, 100)  
/ valign = bottom  
/ halign = left  
</picture>
```

```
<picture topright>  
/ items = ("http://www.millisecond.com/samples/covertattention/rectangle.jpg")  
/ position = (100, 0)  
/ valign = top  
/ halign = right  
</picture>
```

```
<picture bottomright>  
/ items = ("http://www.millisecond.com/samples/covertattention/rectangle.jpg")  
/ position = (100, 100)  
/ valign = bottom  
/ halign = right  
</picture>
```

Those are all the stimuli that we'll present in this script. Next, we'll create the instructions.

2-Creating Instructions

Now lets define how we will present instructions to subjects using the <instruct> element. A script should have only one such element.

```
<instruct>  
/ fontstyle = ("Arial", 18pt, true)  
/ nextlabel = "Press the spacebar to continue"  
/ lastlabel = "Press the spacebar to continue"
```

```
/nextkey = (" ")  
</instruct>
```

We've set the font to 18pt Arial bold. We've also defined the labels that will appear on the buttons that allow subjects to proceed forward through our instruction pages. Finally, we've specified the spacebar key (represented by the space character, " ") as the key to press to advance.

Once we've defined how instruction pages will be presented, we have to create the pages themselves. This part is pretty easy. Note that the "^" character forces a line break. Otherwise, lines of text are word-wrapped.

```
<page inquisit>  
^^The following sample illustrates how to create a covert attention task using Inquisit.  
</page>
```

```
<page intro1>  
^Four boxes will be presented in each corner of the screen, and a fixation point will appear in the center of the screen. Keep your eyes focused on the fixation point throughout the entire experiment.  
^^On each trial, the fixation point will change to an arrow pointing left or right. On 80% of the trials, the arrow points to the side of the screen on which one of the boxes will brighten. On the remaining 20% of the trials, the arrow points in the opposite direction.  
</page>
```

```
<page intro2>  
^Your task is to focus on the fixation point in the center of the screen and indicate whether a box brightened on the left or right side of the screen. If a box on the left brightens, hit the "A" key. If a box on the right brightens, hit the "S" key.  
^^Remember: the arrow will usually point in the direction of the screen with the brighter box, so it is to your advantage to focus on the center fixation point.  
^^Press the space bar to begin practicing the task.  
</page>
```

```
<page begin>  
^^Practice is now complete. Press the space bar to begin the task.  
</page>
```

```
<page finish>  
^^Thank you for participating. The demo is now finished.  
</page>
```

Last, we'll define a special instruction page that we'll use to report performance feedback to our subjects.


```
<page performance>
```

```
^Performance Summary:
```

```
^^You gave the correct answer on <% block.covertattention.percentcorrect %> percent of the trials.
```

```
^^Your average response time was <% block.covertattention.meanlatency %> milliseconds.
```

```
</page>
```

Note the commands "<% block.covertattention.percentcorrect %>" and "<% block.covertattention.meanlatency %>" that appear in the page. These commands will be replaced by the percent correct and mean response latency for all trials in the block called "coverattention", which we will define a little bit later. Inquisit allows you to present a variety of accuracy and latency data to the subject in this way. See help on the [page](#) element for more information.

3-Creating Trials

Now it's time to define the trials for the covert attention task. The trial element specifies which stimuli should be presented, when they are presented, and how the subject should respond. The trial element brings all of the pieces together into a task.

First, we'll define a set of practice trials for the spatial judgment task. The practice trials present our instructions text stimuli that remind the subject how to perform the task.

```
<trial topleftpractice>
```

```
/ stimulustimes = [0=fixation; 500=leftarrow; 700=toplefttarget, bottomleft, topright, bottomright; 1000=instructleft, instructright]
```

```
/ correctresponse = ("a")
```

```
/ validresponse = ("s", "a")
```

```
/ responsetime = 700
```

```
</trial>
```

The "stimulustime" command defines the sequence of stimuli to be presented. The trials presents the fixation point at the beginning of the trial, followed 500 milliseconds later by the left arrow picture. After another 200 milliseconds, the trial presents the target stimulus in the topleft corner and distractor stimuli in other corners. Finally, 300 milliseconds later, the instruction text is presented.

On this trial, the subject can respond by pressing either the "s" or "a" key as defined by the validresponse command. A response of "s" is considered correct as defined by the correctresponse command.

Finally, the `responsetime` command indicates that Inquisit should start measuring the subject's response 700 milliseconds into the stimulus sequence. This corresponds exactly to the time at which the target and distractor stimuli are presented. Response latencies will be reported relative to this point in time. Responses given before this point are ignored.

The remaining practice trials are similar, except that they present the target in different screen locations and therefore. Depending on whether the target is presented on the left or right side of the screen, either "s" or "a" is defined as the correct response.

```
<trial bottomleftpractice>
/ stimulustimes = [0=fixation; 500=leftarrow; 700=topleft, bottomlefttarget, topright,
bottomright; 1000=instructleft, instructright]
/ correctresponse = ("a")
/ validresponse = ("s", "a")
/ responsetime = 700
</trial>
```

```
<trial toprightpractice>
/ stimulustimes = [0=fixation; 500=rightarrow; 700=topleft, bottomleft, toprighttarget,
bottomright; 1000=instructleft, instructright]
/ correctresponse = ("s")
/ validresponse = ("s", "a")
/ responsetime = 700
</trial>
```

```
<trial bottomrightpractice>
/ stimulustimes = [0=fixation; 500=rightarrow; 700=topleft, bottomleft, topright,
bottomrighttarget; 1000=instructleft, instructright]
/ correctresponse = ("s")
/ validresponse = ("s", "a")
/ responsetime = 700
</trial>
```

Now it's time to define the data collection trials. First, we'll define the congruent trials in which the arrow points in the same direction as the target.

```
<trial topleftcongruent>
/ stimulustimes = [0=fixation; 500=leftarrow; 700=toplefttarget, bottomleft, topright,
bottomright]
/ correctresponse = ("a")
/ validresponse = ("s", "a")
/ responsetime = 700
</trial>
```

This trial presents the fixation point followed by the left arrow. The target is presented in the top left corner and the distractors in the remaining corners. Since this is a test trial, we no longer present the instruction reminder stimuli. The following three trials are the same except that the target is presented in the other three corners respectively.

```
<trial bottomleftcongruent>
/ stimulustimes = [0=fixation; 500=leftarrow; 700=topleft, bottomlefttarget, topright,
bottomright]
/ correctresponse = ("a")
/ validresponse = ("s", "a")
/ responsetime = 700
</trial>
```

```
<trial toprightcongruent>
/ stimulustimes = [0=fixation; 500=rightarrow; 700=topleft, bottomleft, toprighttarget,
bottomright]
/ correctresponse = ("s")
/ validresponse = ("s", "a")
/ responsetime = 700
</trial>
```

```
<trial bottomrightcongruent>
/ stimulustimes = [0=fixation; 500=rightarrow; 700=topleft, bottomleft, topright,
bottomrighttarget]
/ correctresponse = ("s")
/ validresponse = ("s", "a")
/ responsetime = 700
</trial>
```

Notice that when the target is in the upper or lower left corner, the correctresponse is defined as the "a" key. When the target is in the upper or lower right, the correctresponse is the "s" key. By including both "a" and "s" in the validresponse command, the experiment will recognize either key press as a response to the trial. All other key presses are ignored.

Finally, we'll define the four types of incongruent trials.

```
<trial topleftincongruent>
/ stimulustimes = [0=fixation; 500=rightarrow; 700=toplefttarget, bottomleft, topright,
bottomright]
/ correctresponse = ("a")
/ validresponse = ("s", "a")
/ responsetime = 700
</trial>
```

```
<trial bottomleftincongruent>
/ stimulustimes = [0=fixation; 500=rightarrow; 700=topleft, bottomlefttarget, topright,
bottomright]
/ correctresponse = ("a")
/ validresponse = ("s", "a")
/ responsetime = 700
</trial>
```

```
<trial toprightincongruent>
/ stimulustimes = [0=fixation; 500=leftarrow; 700=topleft, bottomleft, toprighttarget,
bottomright]
/ correctresponse = ("s")
/ validresponse = ("s", "a")
/ responsetime = 700
</trial>
```

```
<trial bottomrightincongruent>
/ stimulustimes = [0=fixation; 500=leftarrow; 700=topleft, bottomleft, topright,
bottomrighttarget]
/ correctresponse = ("s")
/ validresponse = ("s", "a")
/ responsetime = 700
</trial>
```

We're finished with the trials. Now let's define the blocks.

4-Creating Blocks

The next step is to define the different kinds of blocks that will be used in the experiment. For this experiment, we will define 2 blocks, 1 to run the practice trials and 1 for data collection.

First, let's define the practice block element.

```
<block covertattentionpractice>
/ trials = [1-4 = noreplace(topleftpractice, bottomleftpractice, topleftpractice,
bottomrightpractice)]
/ bgstim = (fixation)
</block>
```

This block element, named *covertattentionpractice*, will run just 4 trials randomly selected from the 4 practice trial types. We've specified that the block should randomly select from the 4 trials without replacement.

Since we are only running 4 trials and there are 4 to select from, that means that each type of trial will be run exactly one time in the block, and the order in which the 4 types are selected will randomly vary.

Now, lets define the data collection block:

```
<block covertattention>
/ preinstructions = (begin)
/ trials = [1-20 = noreplace(topleftcongruent, bottomleftcongruent, topleftcongruent,
bottomrightcongruent,
                                topleftcongruent, bottomleftcongruent, topleftcongruent,
bottomrightcongruent,
                                topleftcongruent, bottomleftcongruent, topleftcongruent,
bottomrightcongruent,
                                topleftcongruent, bottomleftcongruent, topleftcongruent,
bottomrightcongruent,
                                topleftincongruent, bottomleftincongruent, topleftincongruent,
bottomrightincongruent)]
/ bgstim = (fixation)
/ postinstructions = (performance)
</block>
```

The block begins by presenting the instruction page named *begin* as specified by the preinstructions command. It then runs a total of 20 trials randomly selected from the list of data collection trials. Finally, after all the trials have been run it presents an instruction page called *performance*.

You may have noticed that the congruent trial types appear 4 times each in the trial list, whereas incongruent trials appear only once. Why did we repeat some of the trials? The answer is that we want 80% of the trials in this block to be congruent and remaining 20% incongruent, so we've created a random selection pool where the proportion of congruent to incongruent trials is 4 to 1. Since we are selecting without replacement, we are guaranteed that the proportion of selected trials will match the proportions in the selection pool. Of the 20 trials, a randomly selected 16 will be congruent and 4 incongruent.

That does it for the blocks. Now let's define the experiment.

5-Creating an Expt

```
<expt>  
/ preinstructions = (inquisit, intro1, intro2)  
/ blocks = [1=covertattentionpractice; 2=covertattention]  
/ postinstructions = (finish)  
</expt>
```

The *expt* element is quite simple. The *expt* begins by showing a series of three instruction pages, *inquisit*, *intro1*, and *intro2*.

Next, it runs our practice block, followed by the data collection block. Each block is run exactly one time.

Finally, it displays a single instruction page called *finish*,

Last of all, we'll customize the format in which the data is saved using the *data* element.

```
<data>  
/ format = tab  
</data>
```

The *data* element allows me to control what data is recorded, the order of data columns, whether or not to include column labels on the first row, and what character should serve as the column delimiter. In this experiment, we'll specify that the columns should be separated by tab characters, which is a standard text data format recognized by any data analysis software, including Excel and SPSS. For everything else, we'll just use the default settings.

That's it. We're done!

Tutorial: Dot Probe Task

This tutorial builds a simple version of the Dot Probe Task, a commonly used measure of attention. Subjects are presented two words, one above the other, and they are instructed to pronounce the upper word. Occasionally, the upper or lower word is replaced by a "!", in which cases subjects are instructed to press the spacebar as quickly as possible. Typically, reaction times to the "!" are shorter when it appears in the upper position because that is where subjects are attending.

This tutorial covers the following Inquisit features:

- voicekey and keyboard responding
- text presentation
- use of "responsetrial" command to link different types of trials

On the following pages, Inquisit commands are printed in blue, and comments are printed in black:

Steps

1. [Creating Stimuli](#)
2. [Creating Instructions](#)
3. [Creating Trials](#)
4. [Creating Blocks](#)
5. [Creating an Experiment](#)

1-Creating Text Stimuli

The first step in building an experiment is to define all of the stimuli. Stimuli include text or pictures to be shown on a given trial, background text that remains on the screen throughout a block of trials, or a feedback text shown to the subject to indicate when to respond and whether their response was correct or incorrect.

First, let's define the pleasant words appearing in the upper and lower positions:

```
<text pleasanttop>  
/ items = pleasant  
/ position = (50%, 40%)  
</text>
```

```
<text pleasantbottom>
/ items = pleasant
/ position = (50%, 60%)
</text>
```

This text element defines two sets of text stimuli, "pleasanttop" and "pleasantbottom", both of which have two attributes defined, *items* and *position* . The items attribute indicates where the text items are defined. In this case, they are defined in an item element named "pleasant" somewhere else in the script (more on this below). The position attribute specifies where on the screen the text should be presented. Both are presented at the 50% horizontal point of screen (i.e., the horizontal center). The pleasanttop stimulus is presented at the vertical 40% mark, which is 10% of the screen width above center. The pleasantbottom is presented 10% below center.

Now, lets define the items for this text element:

```
<item pleasant>
/ 1 = " HONOR  "
/ 2 = " LUCKY  "
/ 3 = " DIAMOND "
/ 4 = " LOYAL  "
/ 5 = " FREEDOM "
/ 6 = " RAINBOW "
/ 7 = " LOVE   "
/ 8 = " HONEST "
/ 9 = " PEACE  "
/10 = " HEAVEN "
</item>
```

This item element is named "pleasant", which matches the name specified in the items attribute of the text element above. The item set consists of ten pleasant words. Note that the words are padded with spaces so that they are all of equal length when presented in a fixed width font.

Now, lets define the rest of the stimulus categories. First, we'll define the unpleasant words:

```
<text unpleasanttop>
/ items = unpleasant
/ position = (50%, 40%)
</text>
<text unpleasantbottom>
/ items = unpleasant
/ position = (50%, 60%)
</text>
```

and the unpleasant items.


```
<item unpleasant>
/ 1 = "    EVIL    "
/ 2 = "    CANCER  "
/ 3 = "    SICKNESS "
/ 4 = "    DISASTER "
/ 5 = "    POVERTY  "
/ 6 = "    VOMIT   "
/ 7 = "    BOMB     "
/ 8 = "    ROTTEN   "
/ 9 = "    ABUSE    "
/10 = "    MURDER   "
</item>
```

It's a good idea to include instruction text that reminds participants how to respond to the various stimuli. We can do this by presenting text on the screen that are shown in the background throughout a block of trials. So, let's create the instruction text stimulus that reminds the subject to pronounce the upper word and press the spacebar if they see '!'.

```
<text taskreminder>
/ items = ("Pronounce the top word and press the spacebar if you see the '!")
/ position = (50, 15)
/ txcolor = (0, 0, 255)
/ fontstyle = ("Courier New", 12pt)
</text>
```

The reminder stimulus is a bit different than the previous stimuli. First, rather than defining the items in a separate element, we've simply listed the single item directly in the attribute. This inline syntax is a convenient way to define small item sets for things like instructions, focus stimuli, and masks. Also, the *position* attribute specifies that the text should be displayed at the top of the screen. Finally, the *txcolor* attribute specifies that the text should be blue rather than the default color black. Colors in Inquisit are specified as a mix of red, green, and blue components; the *txcolor* attribute specifies 0 intensity for red and green components, and the maximum intensity 255 for the blue component, producing a nice blue color.

Now, let's define the target stimuli '!'. To do this, we create two text stimuli, one of which presents the target in the upper position, and the other which presents it in lower position:

```
<text targettop>
/ items = ("    !    ")
/ position = (50%, 40%)
</text>
<text targetbottom>
/ items = ("    !    ")
/ position = (50%, 60%)
</text>
```

Finally, lets define a focus stimuli that will appear in the center of the screen prior to the two words:

```
<text focuspoint>
/items = ("    +    ")
</text>
```

2-Creating Instructions

Now lets define a set of instruction pages that inform the subject how to perform the task. Defining the instruction pages is easy using the page element. First, we'll define a simple welcome page.

```
<page intro>
^^^Dot Probe Task
^^Welcome and thank you for participating in this task.
^^This task requires that you have a working microphone connected to your computer. If you do
not have a microphone, please press Ctrl+Q now to end the script.
</page>
```

Note that the "^" character is used to force a line break. Otherwise, lines of text are word-wrapped. Now we'll define the rest of the instruction pages:

```
<page task>
Dot Probe Task Instructions:^^
On each trial, two words will be displayed. Your task is to pronounce the TOP word as rapidly as
possible while ignoring the BOTTOM word. ^^
Sometimes, one of the words will be replaced by "!". If you see the "!", press the spacebar as
quickly as possible.
</page>
```

```
<page taskreminder>
Reminder: Pronounce the TOP word as rapidly as possible while ignoring the BOTTOM word. ^^
If you see the "!", press the spacebar as quickly as possible.
</page>
```

```
<page end>
The Dot Probe Task is now concluded.
^^This task illustrates the effect of attention on processing visual stimuli. Typically, people
respond to the "!" more quickly when it appears in the top location because that's where they
are focusing their attention.
</page>
```

Finally, we'll specify how participants can navigate through the instruction pages using the `instruct` element. A script should have only one such element.

```
<instruct>
/ nextkey = (" ")
/ lastlabel = ("Press the spacebar to continue")
/ nextlabel = ("Press the spacebar to continue")
/ fontstyle = ("Arial", 16pt)
</instruct>
```

The *nextkey* attribute indicates that participants must press the spacebar key to advance to the next page. The *nextlabel* and *lastlabel* attributes specifies the text to display on the button label for advancing to the next instruction page, or past the last instruction page. Finally, the *fontstyle* attribute specifies that instructions should be presented in a 16pt Arial font.

3-Creating Trials

The next step is to define the different kinds of trials that will be used in the Dot Probe Task. Trial elements control which stimuli are presented and how the subject may respond to those stimuli. There are eight types of trials used in this task depending on which category of word is presented in the upper position, and whether the word is replaced by a "!" or another word.

First, let's define trials that do not replace the words with a '!':

```
<trial pleasant>
/ stimulustimes = [1=focuspoint; 500=pleasanttop, unpleasantbottom]
/ inputdevice = voicekey
</trial>
```

The trial element's name is *pleasant*. On each line of data in the data file corresponding to this type of trial, this trial name is written.

The *stimulustimes* attribute defines the stimulus presentation sequence of the trial. The focus stimulus is presented for 500 milliseconds, after which pleasant and unpleasant words are presented in the upper and lower positions respectively.

The *inputdevice* attribute specifies the type of input expected from the participant. In this case, the *inputdevice* is "voicekey", which means that Inquisit will treat any sound through the microphone as a valid response, regardless of whether the sound was a valid word.

If we cared about whether the spoken response was an actual word, we could have set this parameter to "speech", in which case Inquisit will use a speech recognition engine to analyze the content of what was said.

Next, we'll define a trial elementsimilar to *pleasant*, differing only in the location where the pleasant and unpleasant stimuli are presented. Here's the definition of trials with unpleasant words in the top position:

```
<trial unpleasant>
/ stimulustimes = [1=focuspoint; 500=unpleasanttop, pleasantbottom]
/ inputdevice = voicekey
</trial>
```

Next come the trials in which one of the words is replaced by a '!'.

```
<trial pleasanttargettop>
/ stimulustimes = [1=focuspoint; 500=pleasanttop, unpleasantbottom]
/ inputdevice = voicekey
/ responsetrial = (anyresponse, targettoppleasant)
</trial>
```

This trial is similar to the two trials above, except that it includes the *responsetrial* command. The response trial specifies a follow up trial to run if a particular response is given. In this case, the followup trial is named "targettoppleasant" and the response is any response. So, whenever this trial runs, it is immediately followed by a trial named "targettoppleasant" to be defined below. This follow up trial presents the '!' stimulus and times the spacebar press.

We will now define the other 2 such trials based on whether the pleasant word is in the upper or lower position, and whether the target is in the upper or lower position.

Trials with insect names classified with the "a" key:

```
<trial unpleasanttargettop>
/ stimulustimes = [1=focuspoint; 500=unpleasanttop, pleasantbottom]
/ inputdevice = voicekey
/ responsetrial = (anyresponse, targettopunpleasant)
</trial>
<trial pleasanttargetbottom>
/ stimulustimes = [1=focuspoint; 500=pleasanttop, unpleasantbottom]
/ inputdevice = voicekey
/ responsetrial = (anyresponse, targetbottompleasant)
</trial>
```

```
<trial unpleasanttargbottom>  
/ stimulustimes = [1=focuspoint; 500=unpleasanttop, pleasantbottom]  
/ inputdevice = voicekey  
/ responsetrial = (anyresponse, targbottomunpleasant)  
</trial>
```

Finally, we'll define the actual follow up trials that present the '!' in the upper or lower position. The first two such trials are identical except for the name. Note that they specify "keyboard" as the inputdevice (this is actually the default, so this command is optional), and spacebar is listed as the only valid and correct response.

```
<trial targettoppleasant>  
/ stimulustimes = [1=targettop]  
/ inputdevice = keyboard  
/ correctresponse = (" ")  
</trial>  
<trial targettopunpleasant>  
/ stimulustimes = [1=targettop]  
/ inputdevice = keyboard  
/ correctresponse = (" ")  
</trial>
```

Since these two trials are identical, why did we define two such trials instead of one? The reason is so that we can easily determine in the data file whether the follow up trial was preceded by an unpleasant or pleasant word. Specifically, the "targettoppleasant" trial is always run after a pleasant word was presented in the upper position, and the "targettopunpleasant" is run after an unpleasant word was in the upper position. Thus, we can analyze the effect of pleasant vs unpleasant by looking at the trial name rather than what was presented on the previous trial.

Last, we'll define the two trials that present the target '!' in the lower position.

```
<trial targetbottompleasant>  
/ stimulustimes = [1=targetbottom]  
/ inputdevice = keyboard  
/ correctresponse = (" ")  
</trial>  
<trial targetbottomunpleasant>  
/ stimulustimes = [1=targetbottom]  
/ inputdevice = keyboard  
/ correctresponse = (" ")  
</trial>
```

Again, these two trials are identical except for their name, but we can use the name to identify whether the preceding trial presented a pleasant or unpleasant word in the upper position.

4-Creating Blocks

The next step is to define the different kinds of blocks that will be used in the experiment. Blocks represent sequences of trials that can be in random or fixed order. For this experiment, two block elements will be defined, one for practice trials and one for data collection.

First, let's define the practice block element for the task.

```
<block practice>
/ trials = [1-20 = noreplace(pleasant, pleasant, unpleasant, unpleasant, pleasanttargettop,
pleasanttargetbottom, unpleasanttargettop, unpleasanttargetbottom)]
/ bgstim = (taskreminder)
</block>
```

This block element is named "practice". The *trials* attribute specifies that the block runs 20 trials randomly selected without replacement from a set of 8 different trials. You may have noticed that 2 of the trials, "pleasant" and "unpleasant", are listed twice. The reason is that we wanted exactly half of the trials in the block to be followed up with a target '!', and the other half not to have a follow up. As the *trials* attribute is specified, 2 of every 8 trials in the block will be "pleasant", 2 will be "unpleasant", 1 will be "pleasanttargettop", 1 will be pleasanttargetbottom, 1 will be unpleasanttargettop, and 1 will be unpleasanttargetbottom.

The *bgstim* attribute specifies that the "taskreminder" instruction text stimulus is presented on the screen as background.

The nonpractice block below (named "critical") is quite similar:

```
<block critical>
/ preinstructions = (taskreminder)
/ trials = [1-36 = noreplace(pleasant, pleasant, pleasant, pleasant, unpleasant, unpleasant,
unpleasant, unpleasant, pleasanttargettop, pleasanttargetbottom, unpleasanttargettop,
unpleasanttargetbottom)]
</block>
```

One difference is that the critical block presents the "taskreminder" instruction page at the beginning of the block as specified by the "preinstructions" attribute. The other difference is that there are 4 "pleasant" and 4 "unpleasant" trials in the selection pool rather than 2 of each. The proportion of trials with follow up trials is now 4 out of 12, or 33%.

5-Creating an Expt

The next step is to define an `expt` element that defines the flow of blocks in the experiment. The `expt` element is defined as follows:

```
<expt>  
/ preinstructions = (intro, task, taskreminder)  
/ postinstructions = (end)  
/ blocks = [1=practice; 2,3=critical]  
</expt>
```

The `expt` element is simple. The *preinstructions* attribute begins the `expt` by showing subjects the 3 pages of instructions, "intro", "task", and "taskreminder". The *postinstructions* attribute specifies final instruction page named "end" to be displayed at the conclusion of the experiment. The *blocks* attribute specifies that 1 practice block is run followed by 2 critical blocks.

Finally, we'll do a little fine tuning by specifying some default settings using the *defaults* element.

```
<defaults>  
/ fontstyle = ("Courier New", 16pt)  
/ posttrialpause = 500  
</defaults>
```

The *fontstyle* attribute specifies that all stimulus and instruction text should be displayed in a 16pt "Courier New" font. The *posttrialpause* attribute specifies that a 500 ms pause should occur at the end of each trial.

The experiment is now complete! You can run the experiment by selecting the "Run" command on the "Experiment" menu.

Tutorial: Demographic Survey

This tutorial builds a simple demographic survey. The sample demonstrates how to design a survey using different types of questions, including dropdown, radiobutton, and textbox questions. It also shows how to place validation rules on textbox input, and to adjust the font and layout of the survey.

On the following pages, Inquisit commands are printed in blue, and comments are printed in black:

Steps

1. [Creating Survey Questions](#)
2. [Creating More Survey Questions](#)
3. [Creating a Survey Page](#)
4. [Creating a Survey](#)

1-Creating Survey Questions

First, we'll create the questions that make up the survey. For any given question, there are a variety of user interface controls at our disposal that give participants a means of making a response. The choice of control depends on the style of question.

The first question on our survey asks respondents to indicate their sex. This is a multiple choice question with two mutually exclusive options. There are a few different controls we can use for this type of question - specifically radiobuttons, listbox, or dropdown. For this question, we'll use the dropdown control because of its space efficiency.

Here is the syntax for creating this dropdown survey item:

```
<dropdown sex>  
/ caption = "Sex"  
/ options = ("female", "male")  
</dropdown>
```

The type of the element is *dropdown*. When users click the control, the list of response options "drops down", allowing them to click on their chosen option. The name of the item is "sex". We'll refer to this item by its name later on in the tutorial when we specify where the item should appear on the page. The *caption* attribute represents the question or instructions the respondents will see for that item. In this case, that caption is simply "Sex".

Finally, the *options* attribute defines the response choices -- either "male" or "female" -- in the dropdown list. Pretty simple.

The next item will ask for the respondent's age. Since there are 100 or more possible responses to this question, a multiple choice format would be cumbersome. Instead, we'll use the textbox element, which simply allow users to type their age into a textbox (or as some call it, an edit box). Here is the definition of our age item:

```
<textbox age>
/ caption = "Age"
/ mask = positiveinteger
/ range = (7, 110)
</textbox>
```

Note the element type is *textbox* and the name is "age". This time, the caption says "Age". Since respondents occasionally make mistakes when typing, we want to make sure the text they enter is in fact a valid age. The *mask* attribute provides a power tool for constraining the type of input that is allowed. In this case, we've set the mask to "positiveinteger", which means that anything other than a positive integer will be considered invalid. However, we don't want to allow just any positive integer. If the respondent types "1" for example, we know that can't be correct because one year-olds don't typically respond to surveys. Similarly, a response of 230432 is invalid because people don't live that long. So, we'll use the *range* attribute to constrain the range of valid ages to a value from 7 to 110.

Next up is the respondent's ethnicity. Just as before, we'll use a dropdown, specifying a caption and the response choices:

```
<dropdown ethnicity>
/ caption = "Ethnicity"
/ options = ("Hispanic or Latino", "Not Hispanic or Latino", "Unknown")
</dropdown>
```

The next question asks for the respondent's race. This is another multiple choice question, but it's slightly different than the previous ones because we'll want to include an "other" option that allows the respondent to enter a response that doesn't appear in the list of options. The dropdown question doesn't support the "other" option, but the radiobuttons control does, so we'll use that:

```
<radiobuttons race>
/ caption = "Race"
/ options = (
  "American Indian/Alaska Native",
  "East Asian",
  "South Asian",
```

```
"Native Hawaiian or other Pacific Islander",
  "Black or African American",
  "White",
  "More than one race - Black/White")
```

```
/ other = "Other"
</radiobuttons>
```

The radiobuttons* element has *caption* and *option* attributes that serve the same purpose as they do with the dropdown element. However, we've specified another attribute called *other* that tells Inquisit to add an "Other" option to the response choices. That options will include a textbox in which respondents can type their race if it isn't in the list.

*Sidenote: You may be wondering why this control is called "radiobuttons". If you were born before 1970, you might recall that the car radios at the time often had a row of punch buttons for selecting a preset radio station. When you punched in one button, the previously selected button popped out, thus assuring that only one station could be selected at a time. The radiobuttons user interface control functions in a similar way, so the name caught on.

Next, we'll ask for participant's political identity. This, too, is a multiple choice question, so either the dropdown or radiobuttons controls would work just fine. However, since liberal is generally associated with "left" and conservative with "right", we'll use a slider control in order to leverage this common association. With a slider control, users respond by sliding a button along a track until it is in the desired position. The track can offer a near-continuous array of positions, or it can force the button into a fixed number of discrete locations. Positions along the track can be labeled to indicate their meaning. With our slider control, respondents will move the button leftward to indicate increasingly liberal values and rightward to indicate increasingly conservative values. Our slider item is defined as follows:

```
<slider political>
/ caption = "Political Identity"
/ labels = (
  "strongly~nliberal", "moderately~nliberal", "slightly~nliberal",
  "neutral", "slightly~nconservative", "moderately~nconservative",
  "strongly~nconservative")
/ range = (1, 7)
/ slidersize = (60%, 5%)
/ showtooltips = false
</slider>
```

The *caption* attribute has the same function as in the previous items. The *labels* attribute specifies the labels that appear in equal distances from left to right along the slider track. Note that labels include the characters "~n". This is not a typo, it is a special character sequence indicating that Inquisit should insert a line break in that position when displaying the label on the screen.

The *range* attribute defines the number of positions on the track. In this case, there are seven positions whose values range from 1 to 7. Since there are also seven labels, each position will align with each of the labels. Next, we'll use the *slidersize* attribute to define the width and height of the slider so that it is wide enough to accommodate all of the labels. We've set the width to be 60% of the width of the computer screen, which should give it plenty of room. Finally, we've set the *showtooltips* attribute to false, so that the control doesn't display the values of each position in a tooltip as the user moves the button along the slider.

For the respondent's occupation, we'll again use a dropdown control. The dropdown is particularly useful in this case because it allows us to display a large number of options in a small amount of screen space:

```
<dropdown occupation>
/ caption = "Occupation"
/ options = ("Administrative Support - Supervisors", "Administrative Support - Financial Clerks",
"Administrative Support - Information and Records",
"Administrative Support - Recording, Scheduling, Dispatching, Distributing",
"Administrative Support - Secretaries and Assistants",
"Administrative Support - Other Support (data entry, office clerk, proofreaders)",
"Arts/Design/Entertainment/Sports - Art and Design", "Arts/Design/Entertainment/Sports -
Entertainers and Performers",
"Arts/Design/Entertainment/Sports - Media and communication", "Arts/Design/Entertainment/
Sports - Media Equipment workers",
"Business - Business Operations", "Business - Financial Specialists", "Computer/Math - Computer
Specialists",
"Computer/Math - Math Scientists";"Computer/Math - Math Technicians", "Construction/
Extraction - Supervisors",
"Construction/Extraction - Construction Trades";"Construction/Extraction - Helpers, Construction
Trades",
"Construction/Extraction - Extraction (e.g., mining, oil)", "Construction/Extraction - Other",
"Education - Postsecondary Teachers", "Education - Primary, Secondary, and Special Ed Teachers",
"Education - Other teachers and instructors", "Education - Librarians, Curators, Archivists",
"Education - Other education, training, and library occupations", "Education - Student",
"Engineers/Architects - Architects, Surveyors, Cartographers", "Engineers/Architects - Engineers",
"Engineers/Architects - Drafters, Engineering and Mapping Technicians", "Farming, Fishing,
Forestry - Supervisors",
"Farming, Fishing, Forestry - Agriculture";"Farming, Fishing, Forestry - Fishing and Hunting",
"Farming, Fishing, Forestry - Forest, Conservation, Logging", "Farming, Fishing, Forestry - Other",
"Food Service - Supervisors", "Food Service - Cooks and food prep", "Food Service - Servers",
"Food Service - Other food service workers (e.g., dishwasher, host)",
"Healthcare - Diagnosing and Treating Practitioners (MD, Dentist, etc.)")
```

"Healthcare - Technologists and Technicians", "Healthcare - Nursing and Home Health Assistants",
 "Healthcare - Occupational and Physical Therapist Assistants", "Healthcare - Other healthcare support",
 "Homemaker or Parenting", "Legal - Lawyers, Judges, and related workers", "Legal - Legal support workers",
 "Maintenance - Building and Grounds Supervisors", "Maintenance - Building workers"; "Maintenance - Grounds Maintenance",
 "Management - Top Executives", "Management - Advertising, Sales, PR, Marketing"; "Management - Operations Specialists",
 "Management - Other Management Occupations", "Military - Officer and Tactical Leaders/Managers",
 "Military - First-line enlisted supervisor/manager", "Military - enlisted tactical, air/weapons, crew, other",
 "Production - Supervisors", "Production - Assemblers and Fabricators"; "Production - Food processing",
 "Production - Metal and Plastic", "Production - Printers", "Production - Textile, Apparel, Furnishings",
 "Production - Woodworkers", "Production - Plant and System Operators", "Production - Other",
 "Protective Service - Supervisors", "Protective Services - Fire fighting and prevention",
 "Protective services - Law Enforcement", "Protective Services - Other (e.g., security, lifeguards, crossing guards)",
 "Repair/Installation - Supervisors", "Repair/Installation - Electrical and Electronic",
 "Repair/Installation - Vehicle and Mobile Equipment", "Repair/Installation - Other", "Retired",
 "Sales - Supervisors", "Sales - Retail"; "Sales - Sales Representatives and Services",
 "Sales - Wholesale and Manufacturing", "Sales - Other sales (e.g., telemarketers, real estate)",
 "Science - Life Scientists", "Science - Physical scientists"; "Science - Social Scientists",
 "Science - Life, Physical, Social Science Technicians", "Service and Personal Care - Supervisors",
 "Service and Personal Care - Animal Care"; "Service and Personal Care - Entertainment attendants",
 "Service and Personal Care - Funeral Service", "Service and Personal Care - Personal Appearance",
 "Service and Personal Care - Transportation, Tourism, Lodging",
 "Service and Personal Care - Other service (e.g., child care, fitness)",
 "Social Service - Counselors, Social Workers, Community specialists", "Social Service - Religious Workers",
 "Transportation - Supervisors", "Transportation - Air Transportation"; "Transportation - Motor Vehicle Operators",
 "Transportation - Rail Transport", "Transportation - Water Transport", "Transportation - Material Moving",
 "Transportation - Other", "Unemployed")
 / optionvalues = (
 "43-1000", "43-3000", "43-4000", "43-5000", "43-6000", "43-9000", "27-1000", "27-2000", "27-3000",
 "27-4000", "13-1000", "13-2000", "15-1000", "15-2000", "15-3000", "47-1000", "47-2000", "47-3000",
 "47-5000", "47-4000", "25-1000", "25-2000", "25-3000", "25-4000", "25-9000", "25-9999", "17-1000",

```
"17-2000", "17-3000", "45-1000", "45-2000", "45-3000", "45-4000", "45-9000", "35-1000", "35-2000",
"35-3000", "35-9000", "29-1000", "29-2000", "31-1000", "31-2000", "31-9000", "00-0000", "23-1000",
"23-2000", "37-1000", "37-2000", "37-3000", "11-0000", "11-2000", "11-3000", "11-9000", "55-1000",
"55-2000", "55-3000", "51-1000", "51-2000", "51-3000", "51-4000", "51-5000", "51-6000", "51-7000",
"51-8000", "51-9000", "33-1000", "33-2000", "33-3000", "33-9000", "49-1000", "49-2000", "49-3000",
"49-9000", "99-0001", "41-1000", "41-2000", "41-3000", "41-4000", "41-9000", "19-1000", "19-2000",
"19-3000", "19-4000", "39-1000", "39-2000", "39-3000", "39-4000", "39-5000", "39-6000", "39-9000",
"21-1000", "21-2000", "53-1000", "53-2000", "53-3000", "53-4000", "53-5000", "53-7000", "53-6000",
"99-9999")
</dropdown>
```

As you can see, there are a lot of choices in the list! You may have also noticed that we're using the *optionvalues* attribute. By default, Inquisit records the text of the selected option into the data file. The *optionvalues* attribute allows us to assign alternative values to each option to be used in recording the data. This is handy if you want to use numeric values or codes when analyzing the data rather than the sometimes long strings of text that are displayed for each response choice. In this case, each occupation will be recorded using the Standard Occupational Classification code as defined by the US Department of Labor.

2-Creating More Survey Questions

The next item allows respondents to indicate their religion. Again, we'll use a dropdown given the large number of options.

```
<dropdown religion>
/ caption = "Religious Affiliation"
/ options = (
  "None", "African Methodist Episcopal Church", "African Methodist Episcopal Zion Church",
"Agnostic",
"American Baptist Association", "American Baptist Churches in the U.S.A.",
"Antiochian Orthodox Christian Diocese of North America", "Armenian Apostolic Church of
America",
"Assemblies of God", "Atheist", "Baha'i", "Baptist Bible Fellowship International", "Baptist General
Conference",
"Baptist Missionary Association of America", "Buddhist", "Christian and Missionary Alliance, The",
"Christian Brethren (Plymouth Brethren)", "Christian Church (Disciples of Christ)",
"Christian Churches and Churches of Christ", "Christian Congregation, Inc., The", "Christian
Methodist Episcopal Church",
"Christian Reformed Church in North America", "Church of God in Christ", "Church of God of
Prophecy",
```

"Church of God (Anderson, IN)", "Church of God (Cleveland, TN)", "Church of Jesus Christ of Latter-day Saints, The",
 "Church of the Brethren", "Church of the Nazarene", "Churches of Christ", "Conservative Baptist Association of America",
 "Coptic Orthodox Church", "Cumberland Presbyterian Church", "Eastern Orthodox", "Eastern Orthodox", "Ecumenical",
 "Episcopal Church", "Evangelical Covenant Church, The", "Evangelical Free Church of America, The",
 "Evangelical Lutheran Church in America", "Evangelical Presbyterian Church", "Free Methodist Church of North America",
 "Full Gospel Fellowship of Churches and Ministers Intl", "General Association of General Baptists",
 "General Association of Regular Baptist Churches", "General Conference Mennonite Brethren Churches",
 "Grace Gospel Fellowship", "Greek Orthodox Archdiocese of America", "Hindu", "Independent Fundamental Churches of America",
 "International Church of the Foursquare Gospel", "International Council of Community Churches",
 "International Pentecostal Holiness Church", "Jehovah's Witnesses", "Jewish", "Lutheran Church-Missouri Synod, The",
 "Mennonite Church", "Muslim/Islamic", "National Assoc of Congregational Christian Churches",
 "National Association of Free Will Baptists", "National Baptist Convention of America, Inc.",
 "National Baptist Convention, USA, Inc.", "National Missionary Baptist Convention of America",
 "Old Order Amish Church",
 "Orthodox Church in America", "Pentecostal Assemblies of the World, Inc.", "Pentecostal Church of God",
 "Pentecostal Church of God", "Presbyterian Church in America", "Presbyterian Church (U.S.A.)",
 "Progressive National Baptist Convention, Inc.", "Reformed Church in America", "Religious Society of Friends (Conservative)",
 "Reorganized Church of Jesus Christ of Latter Day Saints", "Roman Catholic Church, The",
 "Romanian Orthodox Episcopate of America, The", "Salvation Army, The", "Serbian Orthodox Church in the U.S.A. and Canada",
 "Seventh-day Adventist Church", "Sikh", "Southern Baptist Convention", "Unitarian Universalist",
 "United Church of Christ",
 "United Methodist Church, The", "Wesleyan Church, The", "Wisconsin Evangelical Lutheran Synod", "Other")

</dropdown>

Next is education level, again using a dropdown.

<dropdown education>

/ caption = "Education"

/ options = ("elementary", "junior high", "some highschool", "high school graduate", "some college",

"associate's degree", "bachelor's degree", "some graduate school", "masters degree", "M.B.A.", "J.D.", "M.D.", "Ph.D.", "other advanced degree")

</dropdown>

The next two items ask for country of citizenship and residence, respectively. The name of each country will be displayed in our dropdown list, as specified by the *options* attribute. The value recorded in the data file will be the two letter postal abbreviation for that country, as specified by the *optionsvalues* attribute.

<dropdown citizenship>

/ caption = "Country/Region of Primary Citizenship"

/ options = (

"U.S.A.", "Afghanistan", "Albania", "Algeria", "American Samoa", "Andorra", "Angola",
 "Anguilla", "Antarctica", "Antigua And Barbuda", "Argentina", "Armenia", "Aruba",
 "Australia", "Austria", "Azerbaijan", "Bahamas, The", "Bahrain", "Bangladesh", "Barbados",
 "Belarus", "Belgium", "Belize", "Benin", "Bermuda", "Bhutan", "Bolivia", "Bosnia and Herzegovina",
 "Botswana", "Bouvet Island", "Brazil", "British Indian Ocean Territory", "Brunei", "Bulgaria",
 "Burkina Faso", "Burundi", "Cambodia", "Cameroon", "Canada", "Cape Verde", "Cayman Islands",
 "Central African Republic", "Chad", "Chile", "China", "Christmas Island", "Cocos (Keeling) Islands",
 "Colombia", "Comoros", "Congo", "Congo, Democratic Republic of the", "Cook Islands", "Costa
 Rica",
 "Cote D'Ivoire (Ivory Coast)", "Croatia (Hrvatska)", "Cuba", "Cyprus", "Czech Republic", "Denmark",
 "Djibouti", "Dominica", "Dominican Republic", "East Timor", "Ecuador", "Egypt", "El Salvador",
 "Equatorial Guinea", "Eritrea", "Estonia", "Ethiopia", "Falkland Islands (Islas Malvinas)",
 "Faroe Islands", "Fiji Islands", "Finland", "France", "French Guiana", "French Polynesia",
 "French Southern Territories", "Gabon", "Gambia, The", "Georgia", "Germany", "Ghana", "Gibraltar",
 "Greece", "Greenland", "Grenada", "Guadeloupe", "Guam", "Guatemala", "Guinea", "Guinea-Bissau",
 "Guyana", "Haiti", "Heard and McDonald Islands", "Honduras", "Hong Kong S.A.R.", "Hungary",
 "Iceland", "India", "Indonesia", "Iran", "Iraq", "Ireland", "Israel", "Italy", "Jamaica", "Japan",
 "Jordan", "Kazakhstan", "Kenya", "Kiribati", "Korea", "Korea, North", "Kuwait", "Kyrgyzstan",
 "Laos", "Latvia", "Lebanon", "Lesotho", "Liberia", "Libya", "Liechtenstein", "Lithuania", "Luxembourg",
 "Macau S.A.R.", "Macedonia, Former Yugoslav Republic of", "Madagascar", "Malawi", "Malaysia",
 "Maldives", "Mali", "Malta", "Marshall Islands", "Martinique", "Mauritania", "Mauritius", "Mayotte",
 "Mexico", "Micronesia", "Moldova", "Monaco", "Mongolia", "Montserrat", "Morocco", "Mozambique",
 "Myanmar", "Namibia", "Nauru", "Nepal", "Netherlands Antilles", "Netherlands, The", "New
 Caledonia",
 "New Zealand", "Nicaragua", "Niger", "Nigeria", "Niue", "Norfolk Island", "Northern Mariana Islands",
 "Norway", "Oman", "Pakistan", "Palau", "Panama", "Papua New Guinea", "Paraguay", "Peru",
 "Philippines", "Pitcairn Island", "Poland", "Portugal", "Puerto Rico", "Qatar", "Reunion",
 "Romania", "Russia", "Rwanda", "Saint Helena", "Saint Kitts And Nevis", "Saint Lucia",
 "Saint Pierre and Miquelon", "Saint Vincent And The Grenadines", "Samoa", "San Marino",
 "Sao Tome and Principe", "Saudi Arabia", "Senegal", "Seychelles", "Sierra Leone", "Singapore",
 "Slovakia", "Slovenia", "Solomon Islands", "Somalia", "South Africa",

"South Georgia And The South Sandwich Islands", "Spain", "Sri Lanka", "Sudan", "Suriname",
 "Svalbard And Jan Mayen Islands", "Swaziland", "Sweden", "Switzerland", "Syria", "Taiwan",
 "Tajikistan",
 "Tanzania", "Thailand", "Togo", "Tokelau", "Tonga", "Trinidad And Tobago", "Tunisia", "Turkey",
 "Turkmenistan", "Turks And Caicos Islands", "Tuvalu", "Uganda", "Ukraine", "United Arab Emirates",
 "United Kingdom", "U.S.A.", "United States Minor Outlying Islands", "Uruguay", "Uzbekistan",
 "Vanuatu", "Vatican City State (Holy See)", "Venezuela", "Vietnam", "Virgin Islands (British)",
 "Virgin Islands (US)", "Wallis And Futuna Islands", "Yemen", "Yugoslavia", "Zambia", "Zimbabwe")

/ optionvalues = (

"US", "AF", "AL", "DZ", "AS", "AD", "AO", "AI", "AQ", "AG", "AR", "AM", "AW", "AU", "AT", "AZ", "BS",
 "BH", "BD", "BB", "BY", "BE", "BZ", "BJ", "BM", "BT", "BO", "BA", "BW", "BV", "BR", "IO", "BN", "BG",
 "BF", "BI", "KH", "CM", "CA", "CV", "KY", "CF", "TD", "CL", "CN", "CX", "CC", "CO", "KM", "CG", "CD",
 "CK", "CR", "CI", "HR", "CU", "CY", "CZ", "DK", "DJ", "DM", "DO", "TP", "EC", "EG", "SV", "GQ", "ER",
 "EE", "ET", "FK", "FO", "FJ", "FI", "FR", "GF", "PF", "TF", "GA", "GM", "GE", "DE", "GH", "GI", "GR",
 "GL", "GD", "GP", "GU", "GT", "GN", "GW", "GY", "HT", "HM", "HN", "HK", "HU", "IS", "IN", "ID", "IR",
 "IQ", "IE", "IL", "IT", "JM", "JP", "JO", "KZ", "KE", "KI", "KR", "KP", "KW", "KG", "LA", "LV", "LB",
 "LS", "LR", "LY", "LI", "LT", "LU", "MO", "MK", "MG", "MW", "MY", "MV", "ML", "MT", "MH", "MQ", "MR",
 "MU", "YT", "MX", "FM", "MD", "MC", "MN", "MS", "MA", "MZ", "MM", "NA", "NR", "NP", "AN", "NL", "NC",
 "NZ", "NI", "NE", "NG", "NU", "NF", "MP", "NO", "OM", "PK", "PW", "PA", "PG", "PY", "PE", "PH", "PN",
 "PL", "PT", "PR", "QA", "RE", "RO", "RU", "RW", "SH", "KN", "LC", "PM", "VC", "WS", "SM", "ST", "SA",
 "SN", "SC", "SL", "SG", "SK", "SI", "SB", "SO", "ZA", "GS", "ES", "LK", "SD", "SR", "SJ", "SZ", "SE",
 "CH", "SY", "TW", "TJ", "TZ", "TH", "TG", "TK", "TO", "TT", "TN", "TR", "TM", "TC", "TV", "UG", "UA",
 "AE", "UK", "US", "UM", "UY", "UZ", "VU", "VA", "VE", "VN", "VG", "VI", "WF", "YE", "YU", "ZM", "ZW")

</dropdown>

<dropdown residence>

/ caption = "Country/Region of Residence"

"U.S.A.", "Afghanistan", "Albania", "Algeria", "American Samoa", "Andorra", "Angola",
 "Anguilla", "Antarctica", "Antigua And Barbuda", "Argentina", "Armenia", "Aruba",
 "Australia", "Austria", "Azerbaijan", "Bahamas, The", "Bahrain", "Bangladesh", "Barbados",
 "Belarus", "Belgium", "Belize", "Benin", "Bermuda", "Bhutan", "Bolivia", "Bosnia and Herzegovina",
 "Botswana", "Bouvet Island", "Brazil", "British Indian Ocean Territory", "Brunei", "Bulgaria",
 "Burkina Faso", "Burundi", "Cambodia", "Cameroon", "Canada", "Cape Verde", "Cayman Islands",
 "Central African Republic", "Chad", "Chile", "China", "Christmas Island", "Cocos (Keeling) Islands",
 "Colombia", "Comoros", "Congo", "Congo, Democratic Republic of the", "Cook Islands", "Costa
 Rica",
 "Cote D'Ivoire (Ivory Coast)", "Croatia (Hrvatska)", "Cuba", "Cyprus", "Czech Republic", "Denmark",
 "Djibouti", "Dominica", "Dominican Republic", "East Timor", "Ecuador", "Egypt", "El Salvador",
 "Equatorial Guinea", "Eritrea", "Estonia", "Ethiopia", "Falkland Islands (Islas Malvinas)",
 "Faroe Islands", "Fiji Islands", "Finland", "France", "French Guiana", "French Polynesia",
 "French Southern Territories", "Gabon", "Gambia, The", "Georgia", "Germany", "Ghana", "Gibraltar",
 "Greece", "Greenland", "Grenada", "Guadeloupe", "Guam", "Guatemala", "Guinea", "Guinea-Bissau",

"Guyana", "Haiti", "Heard and McDonald Islands", "Honduras", "Hong Kong S.A.R.", "Hungary",
 "Iceland", "India", "Indonesia", "Iran", "Iraq", "Ireland", "Israel", "Italy", "Jamaica", "Japan",
 "Jordan", "Kazakhstan", "Kenya", "Kiribati", "Korea", "Korea, North", "Kuwait", "Kyrgyzstan",
 "Laos", "Latvia", "Lebanon", "Lesotho", "Liberia", "Libya", "Liechtenstein", "Lithuania", "Luxembourg",
 "Macau S.A.R.", "Macedonia, Former Yugoslav Republic of", "Madagascar", "Malawi", "Malaysia",
 "Maldives", "Mali", "Malta", "Marshall Islands", "Martinique", "Mauritania", "Mauritius", "Mayotte",
 "Mexico", "Micronesia", "Moldova", "Monaco", "Mongolia", "Montserrat", "Morocco", "Mozambique",
 "Myanmar", "Namibia", "Nauru", "Nepal", "Netherlands Antilles", "Netherlands, The", "New
 Caledonia",
 "New Zealand", "Nicaragua", "Niger", "Nigeria", "Niue", "Norfolk Island", "Northern Mariana Islands",
 "Norway", "Oman", "Pakistan", "Palau", "Panama", "Papua New Guinea", "Paraguay", "Peru",
 "Philippines", "Pitcairn Island", "Poland", "Portugal", "Puerto Rico", "Qatar", "Reunion",
 "Romania", "Russia", "Rwanda", "Saint Helena", "Saint Kitts And Nevis", "Saint Lucia",
 "Saint Pierre and Miquelon", "Saint Vincent And The Grenadines", "Samoa", "San Marino",
 "Sao Tome and Principe", "Saudi Arabia", "Senegal", "Seychelles", "Sierra Leone", "Singapore",
 "Slovakia", "Slovenia", "Solomon Islands", "Somalia", "South Africa",
 "South Georgia And The South Sandwich Islands", "Spain", "Sri Lanka", "Sudan", "Suriname",
 "Svalbard And Jan Mayen Islands", "Swaziland", "Sweden", "Switzerland", "Syria", "Taiwan",
 "Tajikistan",
 "Tanzania", "Thailand", "Togo", "Tokelau", "Tonga", "Trinidad And Tobago", "Tunisia", "Turkey",
 "Turkmenistan", "Turks And Caicos Islands", "Tuvalu", "Uganda", "Ukraine", "United Arab Emirates",
 "United Kingdom", "U.S.A.", "United States Minor Outlying Islands", "Uruguay", "Uzbekistan",
 "Vanuatu", "Vatican City State (Holy See)", "Venezuela", "Vietnam", "Virgin Islands (British)",
 "Virgin Islands (US)", "Wallis And Futuna Islands", "Yemen", "Yugoslavia", "Zambia", "Zimbabwe")
 / optionvalues = (
 "US", "AF", "AL", "DZ", "AS", "AD", "AO", "AI", "AQ", "AG", "AR", "AM", "AW", "AU", "AT", "AZ", "BS",
 "BH", "BD", "BB", "BY", "BE", "BZ", "BJ", "BM", "BT", "BO", "BA", "BW", "BV", "BR", "IO", "BN", "BG",
 "BF", "BI", "KH", "CM", "CA", "CV", "KY", "CF", "TD", "CL", "CN", "CX", "CC", "CO", "KM", "CG", "CD",
 "CK", "CR", "CI", "HR", "CU", "CY", "CZ", "DK", "DJ", "DM", "DO", "TP", "EC", "EG", "SV", "GQ", "ER",
 "EE", "ET", "FK", "FO", "FJ", "FI", "FR", "GF", "PF", "TF", "GA", "GM", "GE", "DE", "GH", "GI", "GR",
 "GL", "GD", "GP", "GU", "GT", "GN", "GW", "GY", "HT", "HM", "HN", "HK", "HU", "IS", "IN", "ID", "IR",
 "IQ", "IE", "IL", "IT", "JM", "JP", "JO", "KZ", "KE", "KI", "KR", "KP", "KW", "KG", "LA", "LV", "LB",
 "LS", "LR", "LY", "LI", "LT", "LU", "MO", "MK", "MG", "MW", "MY", "MV", "ML", "MT", "MH", "MQ", "MR",
 "MU", "YT", "MX", "FM", "MD", "MC", "MN", "MS", "MA", "MZ", "MM", "NA", "NR", "NP", "AN", "NL", "NC",
 "NZ", "NI", "NE", "NG", "NU", "NF", "MP", "NO", "OM", "PK", "PW", "PA", "PG", "PY", "PE", "PH", "PN",
 "PL", "PT", "PR", "QA", "RE", "RO", "RU", "RW", "SH", "KN", "LC", "PM", "VC", "WS", "SM", "ST", "SA",
 "SN", "SC", "SL", "SG", "SK", "SI", "SB", "SO", "ZA", "GS", "ES", "LK", "SD", "SR", "SJ", "SZ", "SE",
 "CH", "SY", "TW", "TJ", "TZ", "TH", "TG", "TK", "TO", "TT", "TN", "TR", "TM", "TC", "TV", "UG", "UA",
 "AE", "UK", "US", "UM", "UY", "UZ", "VU", "VA", "VE", "VN", "VG", "VI", "WF", "YE", "YU", "ZM", "ZW")
 </dropdown>

Finally, we'll ask participants to enter their postal code using a textbox:

```
<textbox zipcode>  
/ caption = "Current Postal Code"  
</textbox>
```

Note that Inquisit has a mask called "uszipcode" that constrains the input to be a valid United States zip code (5 digit format, plus 4 optional). If we expect respondents from other countries, however, we should leave off this constraint since postal codes throughout the world come in a variety of formats.

3-Creating Survey Pages

Once you've defined the questions, the next step is to determine how to layout those questions on the pages of the survey. Inquisit allows you break out questions into separate pages/screens. Respondents can answer the questions on a single page and then click the "Next" button to answer more questions on the next page. The survey can be configured to allow them to navigate back to previous page and change their answers, or to allow forward only navigation.

Inquisit allows us to configure the font size of each question along with the spacing between questions. By using small fonts and spacings, we could try to squeeze all of the questions onto a single page. The result would look cramped, however, so we'll separate the questions into two pages.

The first page is specified here:

```
<surveypage demographics1>  
/ caption = "Please answer the following demographic questions"  
/ fontstyle = ("Verdana", -16, true, false, false, 5, 0)  
/ questions = [1=sex; 2=age; 3=ethnicity; 4=race; 5=political; 6=occupation]  
</surveypage>
```

The element type is `surveypage`, and the name of this page "demographics1". The *caption* attribute tells Inquisit to display a simple instruction at the top of the page. The *fontstyle* attribute specifies the font in which caption appears, 14pt Verdana in bold. Finally, the *questions* attribute lists the items that we defined previously that should be included on the page. In this case, we've specified six questions -- sex, age, ethnicity, race, political, and occupation -- presented in that order.

Now let's define the second page:

```
<surveypage demographics2>
/ caption = "Please answer the following demographic questions (continued)"
/ fontstyle = ("Verdana", -16, true, false, false, false, 5, 0)
/ questions = [1=religion; 2=education; 3=citizenship; 4=residence; 5=zipcode]
</surveypage>
```

This page has a caption similar to the first page, although we've added "(continued)" to it. The font style is the same as the first page. This page presents the remaining 5 questions that we previously created -- religion, education, citizenship, residence, and zipcode -- in that order.

That wasn't too painful.

4-Creating the Survey

Last but not least, we need to specify the order in which the pages are presented, along with some other global settings. For this, we use the *survey* element:

```
<survey demographics>
/ pages = [1=demographics1; 2=demographics2]
/ responsefontstyle = ("Verdana", -12, false, false, false, false, 5, 0)
/ itemfontstyle = ("Verdana", -13, false, false, false, false, 5, 0)
/ itemspacing = 2%
/ showpagenumbers = false
</survey>
```

We've named our survey "demographics". The *pages* attribute specifies which pages appear in the survey and in what order. Our survey presents the surveypage called "demographics1" followed by the surveypage called "demographics2".

The *responsefontstyle* attribute specifies the font to use for response options in the survey. This font applies to the choices in our multiple choice items, and the text that the user types in the textbox items.

The *itemfontstyle* attribute specifies the font used for the captions of our survey questions.

The *itemspacing* attribute specifies that the space between questions should be 2% of the height of the screen.

The *showpagenumbers* attribute specifies whether each page should be numbered. Since our survey is only two pages, we'll turn off page numbering by setting this attribute to false.

That's it - our survey is complete. You can run the survey by selecting the "Run" command on Inquisit's "Experiment" menu. Once you've completed the survey, the data will be recorded into a file called "demographics.dat". The name based on the name of the survey element. The data file will be located in the same folder as your script file.

Inquisit How To's

How to Run an Experiment

There are several ways to run an Inquisit experiment.

Using Inquisit's Menu

- Launch Inquisit from the Windows Start Menu.
- Select the Open command on Inquisit's File menu.
- Choose the script file you would like to run.
- Select the Run command on Inquisit's Experiment menu.
- When prompted, enter a subject number and press the OK button.

Using Windows Explorer

- Find the script file you would like to run in Windows Explorer.
 - Double click on the script file.
 - When prompted, enter a subject number and press the OK button.
- (You can also create shortcuts to Inquisit scripts, and launch the script by double clicking on the shortcut icon.)

Using the Windows Command Prompt

For details on running Inquisit from a command prompt, see [Running an Experiment from the Command Line](#)

Interrupting an Experiment or Block in Progress

While the experiment is running,

- Press Ctrl+b to skip the rest of the trials of the current block.
- Press Ctrl+q to immediately end the experiment.

How to Run an Experiment from the Command Line

Inquisit allows you to launch an experiment using command lines. Command lines can be entered, for example, using the Run option from the Windows Start Menu. Inquisit uses standard Windows command line rules, but provides the additional option of specifying a subject number.

The syntax for command lines is:

```
C:\>"inquisitpath" "scriptpath" subjectnumber monkey|human "password"
```

where the parameters are as follows:

inquisitpath

Quoted, fully qualified path to the Inquisit.exe file. For example, "C:\Program Files\Millisecond Software\Inquisit 3\Inquisit.exe".

scriptpath

Quoted, fully qualified path to the script file to run. For example, "C:\My Scripts\script.exp".

subjectnumber

A positive integer specifying the subject number for this run.

monkey|human

If monkey is specified, the automated monkey runs the script. Otherwise, if human is specified, the script runs in standard interactive mode.

password

Specifies the password use to decrypt an encrypted script file. The password must exactly match the password given when the encrypted file was saved.

Only the full path to Inquisit.exe is required, which alone will simply launch Inquisit.

If the full path to a script file is included in the command, Inquisit is launched, the script file is opened, the user is prompted for a subject number, and the experiment begins.

If the subject number is also included in the command, Inquisit is launched and the script is run using the specified subject number. The subject number option is only valid when a script file is specified.

How to Run Blocks, Trials, and Stimuli Using the object browser

The object browser lets you inspect elements of your experiment without having to run the entire script. You can browse specific stimuli and run specific trials or blocks. The object browser is a great tool for demonstrating tasks and methods in lectures and presentations, and provides a convenient way to quickly test new elements.

To use the object browser:

1. Select the "Object Browser" command from the Tools menu.
2. On the object browser dialog, select the tab for the type of object - Page, Stimulus, Trial, or Block - you would like to observe.
3. Select the specific instruction page, stimulus, trial, block you would like to observe from the list.
4. Click the "Do" button.

How to Present Stimuli Provided by Subjects

Inquisit supports defining stimulus sets based on input provided by the subject. This feature can be used, for example, to present text provided by the subject (e.g., the subject's stated name), or to present only stimuli selected by the subject on previous trials.

Presenting text entered by the subject

To create a set of items consisting of text entered by the subject, first we'll define an empty item set to store the items:

```
<item responseitems>  
</item>
```

Next, we'll create a survey page with textbox items for gathering the respondent's first and last name:

```
<textbox firstname>  
/ caption = "Please enter your first name:"  
</textbox>
```

```
<textbox lastname>  
/ caption = "Please enter your last name:"  
</textbox>
```

```
<surveypage page1>  
/ questions = [1=firstname; 2=lastname]  
/ ontrialend = [item.responseitems.item = textbox.firstname.response]  
/ ontrialend = [item.responseitems.item = textbox.lastname.response]  
</surveypage>
```

This is a survey page with two items allowing respondents to enter their first and last names. The ontrialend attributes contain the commands that actually do the work of adding the names to the responseitems set. The first ontrialend command adds a new item to responseitems by setting the item property to the response for the firstname question.

The second `ontrialend` command adds the last name in a similar fashion. Neither of these commands changes or removes the items that are already in the `responseitems` item set - they simply add new items. Using this technique, we can build up item sets of arbitrary length containing user-supplied content.

If we later want to change the first name item to something else (e.g., a nickname), we could do so using the following command:

```
<surveypage page2>  
/ questions = [1=nickname]  
/ ontrialend = [item.responseitems.item.1 = textbox.nickname.response]  
</surveypage>
```

This time, the `ontrialend` command assigns the first item in the `responseitems` set to the middle name. This time, the command includes the index number "1", which refers specifically to the first item in the set. By specifying the index number, we can thus change item in that position. If the index number is omitted as in the first example, a new item is added.

Stimulus elements can use the `responseitems` element the same way they would use other item elements. For example:

```
<text responses>  
/ txcolor = (0, 255, 0)  
/ position = (25, 25)  
/ items = responseitems  
</text>
```

This text element presents the `responseitems` in the upper left corner of the screen in green.

Text items selected by the participant

In addition to allowing participants to type in new stimuli, it is also possible for them to choose items from a predetermined list. Imagine we want to create an item set containing three cities the participant has never visited. We'll store these items in an item set called "cities":

```
<item cities>  
</item>
```

Next, we'll create a survey page with a checkboxes item with a list of cities for the participant to choose from:

```
<checkboxes cities>  
/ caption = "Select three of the cities below that you have never visited:"  
/ options = ("Berlin", "London", "Tokyo", "Paris", "Rome", "Sydney", "New York")  
/ range = (3, 3)  
</checkboxes>
```



```
<surveypage page1>
/ questions = [1=cities]
/ ontrialend = [ if (checkboxes.cities.checked.1 == true) item.cities.item =
checkboxes.cities.option.1 ]
/ ontrialend = [ if (checkboxes.cities.checked.2 == true) item.cities.item =
checkboxes.cities.option.2 ]
/ ontrialend = [ if (checkboxes.cities.checked.3 == true) item.cities.item =
checkboxes.cities.option.3 ]
/ ontrialend = [ if (checkboxes.cities.checked.4 == true) item.cities.item =
checkboxes.cities.option.4 ]
/ ontrialend = [ if (checkboxes.cities.checked.5 == true) item.cities.item =
checkboxes.cities.option.5 ]
/ ontrialend = [ if (checkboxes.cities.checked.6 == true) item.cities.item =
checkboxes.cities.option.6 ]
/ ontrialend = [ if (checkboxes.cities.checked.7 == true) item.cities.item =
checkboxes.cities.option.7 ]
</surveypage>
```

The checkboxes item lists seven cities and uses the *range* attribute to constrain the number of selections to 3. There are seven ontrialend commands, each of which evaluates whether its corresponding city is checked, and if so, adds it to the cities item list. The items can then be presented using a text element.

```
<text responses>
/ items = cities
</text>
```

Next, define an item element whose items will consist of the picture or text items selected by the subject on each run of the multiplechoice trial:

```
<item selecteditems>
/ items = (multiplechoice)
</item>
```

Each time a multiplechoice trial is run, Inquisit adds the item selected by the subject to the selecteditems item set. If the multiplechoice trial presented pictures, the selecteditems item element could be presented by picture elements in the script. If the multiplechoice trial presented text, the selecteditem item element could be presented by text elements.

Picture items selected by the participant

We'll use the cities example again, but modify it to create an item set of pictures. Again, we'll store these items in an item set called "cities":

```
<item cities>
</item>
```

Here is our checkboxes survey item with the list of cities. This time, we've added the *optionvalues* attribute to define, for each option, the name of the corresponding jpg file containing a picture of the city. The checkboxes item will still display the city names next to each the checkbox, not the underlying optionsvalues.

```
<checkboxes cities>
/ caption = "Select three of the cities below that you have never visited:"
/ options = ("Berlin", "London", "Tokyo", "Paris", "Rome", "Sydney", "New York")
/ optionvalues = ("berlin.jpg", "london.jpg", "tokyo.jpg", "paris.jpg", "rome.jpg", "sydney.jpg",
"newyork.jpg")
/ range = (3, 3)
</checkboxes>
```

Finally, here is the surveypage that displays the checkboxes item:

```
<surveypage page1>
/ questions = [1=cities]
/ ontrialend = [ if (checkboxes.cities.checked.1 == true) item.cities.item =
checkboxes.cities.optionvalues.1 ]
/ ontrialend = [ if (checkboxes.cities.checked.2 == true) item.cities.item =
checkboxes.cities.optionvalues.2 ]
/ ontrialend = [ if (checkboxes.cities.checked.3 == true) item.cities.item =
checkboxes.cities.optionvalues.3 ]
/ ontrialend = [ if (checkboxes.cities.checked.4 == true) item.cities.item =
checkboxes.cities.optionvalues.4 ]
/ ontrialend = [ if (checkboxes.cities.checked.5 == true) item.cities.item =
checkboxes.cities.optionvalues.5 ]
/ ontrialend = [ if (checkboxes.cities.checked.6 == true) item.cities.item =
checkboxes.cities.optionvalues.6 ]
/ ontrialend = [ if (checkboxes.cities.checked.7 == true) item.cities.item =
checkboxes.cities.optionvalues.7 ]
</surveypage>
```

For each city listed in the checkboxes item, the corresponding ontrialend command evaluates whether its checkbox is checked, and if so, adds the optionvalue (i.e., the picture file name) to item set. Now, we can create a picture element that will display these pictures:

```
<picture selectedcities>
/ items = cities
</picture>
```

One additional step is required to make this solution work. To maximize performance, Inquisit loads all pictures files used by a script at the very beginning as it parses the script. We therefore we need to tell Inquisit that the script uses these files, or it won't load them. We can do this by creating a picture element that lists these files as items. We won't actually use this element anywhere in the script - it's just there so that Inquisit will load the pictures:

```
<picture dummy>  
/ items = ("berlin.jpg", "london.jpg", "tokyo.jpg", "paris.jpg", "rome.jpg", "sydney.jpg",  
"newyork.jpg")  
</item>  
There you have it.
```

How to Present Stimulus Pairs

Often an experiment will require presentation of stimulus pairs. For example, a lexical priming task may have prime-target pairs such DOCTOR/NURSE that are meant to be presented together on a given trial.

Stimulus pairs are defined in Inquisit by creating two stimulus elements for the first and second members of each pair, then linking selection of the second with that of the first. For example:

```
<text firstname>  
/ items = ("BILL", "LINDON")  
/ select = noreplace  
</text>
```

```
<text lastname>  
/ items = ("CLINTON", "JOHNSON")  
/ select = current(firstname)  
</text>
```

For `firstname`, `select = noreplace` means that whenever `firstname` is presented, it randomly selects without replacement which specific item is shown. For `lastname`, `select = current(firstname)` indicates that whenever a `lastname` is presented, it selects the item corresponding to the currently selected `firstname` item.

The trial below presents both stimuli:

```
<trial person>  
/ stimulustimes = [0 = firstname; 100 = lastname]  
/ response = anyresponse  
</text>
```

On each run of this trial, BILL is always followed by CLINTON and LINDON is always followed by JOHNSON.

How to Test an Experiment

Inquisit has a built in 'monkey' who will happily serve as a test subject for even your longest, most tedious experiments. The monkey is useful for debugging and generating sample data. To run the monkey:

- a) Open the experiment script in Inquisit.
- b) Select the "Monkey" command from the Tools menu.
- c) Enter a subject number when prompted and then press OK.

Now, go enjoy a cup of coffee as the monkey runs through the entire experiment without so much as a complaint. Consult your animal subjects ethics board to determine whether the monkey's informed consent is required.

How to Interrupt an Experiment

While the experiment is running and waiting for a response,

- Press Ctrl+b to skip the rest of the trials of the current block.
- Press Ctrl+q to immediately end the experiment.

How to Audit Stimulus Presentation Times

Inquisit uses DirectX to present stimuli as rapidly as is possible using Windows. However, even the most ingeniously architected, most efficient software can not guarantee accurate timing of stimuli presentations on Windows *every time*. Alas, Windows was not designed to be a real time operating system, and certain factors may potentially introduce anomalies into stimulus presentation times.

For example, an experiment might present an unusually large number of high resolution, high color images on a single trial, the combination of which exceeds display memory capacity. Or, a high priority system thread may steal CPU cycles from Inquisit while it is in the middle of a stimulus presentation sequence.

To be sure, the experimenter can minimize the chances of the former situation by making sure the system has enough display memory to run her task paradigm. Furthermore, Inquisit minimizes the chances that other programs, including Windows itself, will steal CPU cycles from it by setting its process and threads to the highest priority level. It should also be noted that tests of Inquisit on a variety of hardware configurations have shown that timing anomalies are *exceedingly* rare. The point, however, is that on a system as complex as a Windows PC, it is impossible to guarantee such anomalies will *never* occur.

For this reason, Inquisit includes self-auditing capabilities that allow the researcher to determine when a stimulus was actually presented. The actual stimulus onset times may be recorded directly into the data file, where they can be casually inspected or incorporated into data analysis, for example, as a criterion for discarding trials.

To record stimulus onset times, specify the "stimulusonset" keyword in the [columns](#) attribute of the [data](#) element:

```
<data>  
/ columns = [stimulusonset stimulusonset stimulusonset stimulusonset]  
</data>
```

The above example writes the stimulus onset time, starting from the beginning of the stimulus presentation sequence, of the first, second, third, and fourth stimulus presented on each trial of the experiment. If less than four stimuli are presented on a given trial, a '0' is recorded for the extras.

How to Erase Stimuli

By default, Inquisit erases all stimuli presented on a given trial after the subject has responded. Note that you can prevent a stimulus from being erased at the end of a trial by setting its [erase](#) attribute to *false*.

Some research tasks require that stimuli be presented for a fixed duration, after which it should be removed from the screen. To erase a stimulus after a fixed duration, you must create a separate 'blank' stimulus element and then present that blank at the appropriate interval in the stimulus presentation sequence so that it overwrites the stimulus you wish to erase.

There are a number of ways to create a blank stimulus. The easiest is to create a rectangular [shape](#) element, set its size large enough to cover the stimulus you wish to erase, and then set its color to the background color of the screen.

If you are erasing text items of various sizes, it may make sense to define a second text element that uses the same set of items, sets the [select](#) attribute as linked with the to-be-erased stimulus, but sets the [txcolor](#) and [txbgcolor](#) attributes to the background color of the screen. Each time this blank text stimulus is presented, it will present a erasing rectangle that is exactly the same size of the original text stimulus.

How to Analyze Recorded Voice Responses

Many reaction time tasks are best administered by requiring participants to make verbal responses. Consider the classic Stroop task, for example, whereby the respondent is required to identify the color of the ink in which color names are printed. Although it's possible to demonstrate the Stroop effect using key or button presses, the effect is much more evident with verbal responses. Unfortunately, analyzing the accuracy of verbal responses typically requires the tedious and time-consuming process of transcribing hundreds or even thousands of recorded vocalizations. Given the amount of labor involved, many researchers avoid verbal response tasks altogether in favor of procedures that can be completely computer automated.

Inquisit removes the tedium of transcribing verbal responses by leveraging speech recognition software. With Inquisit, spoken responses can be automatically analyzed in real time, or you can record responses and then later analyze them with the speech engine.

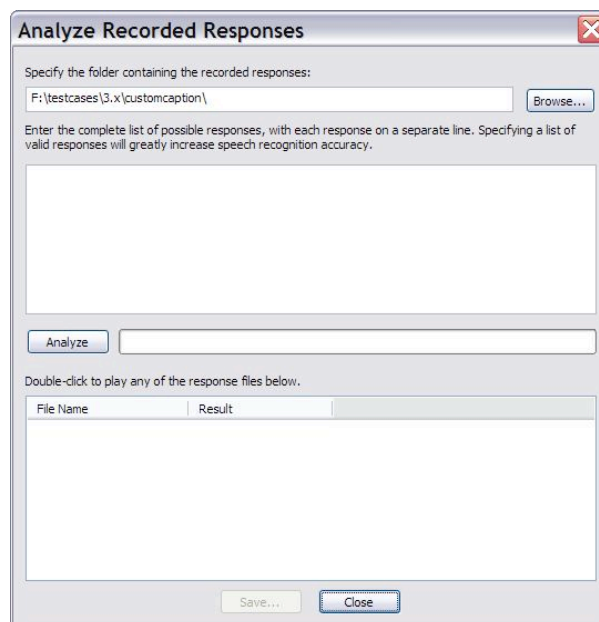
To analyze the content of spoken responses in real time, simply set the [inputdevice](#) attribute to *speech*.

With this option, Inquisit activates the speech engine on every voice trial, listens for a response, and then attempts to identify the utterance from a list of possible responses that you specify. The identified response is recorded directly into the data file. In cases where the response could not be identified, Inquisit records a "?" instead. The advantages of this option are that a) the utterance is immediately available in the data file for analysis, and b) Inquisit can determine whether a correct or incorrect response was given for purposes of providing tracking performance and providing error feedback. The disadvantages of *speech* option are that the speech engine may occasionally misidentify or fail to recognize valid utterances. It also may impose a perceptible delay between the time an utterance is made and the time it is recognized (importantly, this delay does not affect the measurement of response latency of spoken responses).

To record responses for subsequent speech recognition analysis, set the [inputdevice](#) attribute to *voicerecord*. With this option, Inquisit listens for a response on every trial, and when it picks up incoming sound, that sound is recorded to a wav file. Separate wave files are recorded for each trial, and files are named in a way that allows you to match the file to the particular trial of the particular session for the particular subject that made the response. When you are ready to analyze the data, Inquisit provides a handy tool that "listens" to each wav file, identifies the spoken content, and saves the results to a tab delimited file.

The "Analyze Recorded Responses" Tool

To analyze the recorded responses, click Inquisit's Tools menu, and select the "Analyze Recorded Responses..." command. This will open the following window:



First, you must specify the folder containing the recorded wav files. You will find these files in a subfolder called "voicerecord" located in the folder containing your script file.

Next, you can optionally specify the complete list of valid utterances so that the engine knows what words to listen for. If your task has a fixed set of valid responses, specifying them here will *greatly* improve the recognition accuracy of the engine. If you do not specify valid responses, the engine treats the entire lexicon as a potentially valid response, and recognition accuracy suffers accordingly.

To start the analysis, click the Analyze button. Once all the files have been analyzed, each wav file and its recognition result appear in the list below. If the engine could not identify the response, a "?" appears. To listen to any of the wav files, simply double-click the file in the results list and it will play (make sure audio is configured correctly, speakers are turned on, and the volume is turned up). By listening to the files, you can double-check the engine's accuracy or try to identify an utterance that the engine could not.

Finally, you can save the results to a tab-delimited file by clicking the "Save..." button. The file will contain two columns of data for the file name and recognition result respectively. The saved data can then be inserted into the main data file using the command language and macros of your stats software, or using good old fashioned Filter, Sort, Copy, and Paste with a spreadsheet program like Excel.

How to Adjust the Response Window

The temporal characteristics of the response window may be adjusted from block to block depending on a given subject's performance. The way the script specifies the response window method also determines the adjustment procedure.

The default window adjustment procedure is:

The response window is moved back 33 ms for subsequent blocks if

- a. the percent of correct responses for the block $\leq 55\%$.
- b. the percent of correct responses for the block $\leq 65\%$ and mean latency is over 100 ms greater than the current window center.

The response window is moved forward 33 ms for subsequent blocks if

- a. the percent of correct responses for the block $\geq 80\%$ and mean latency is no more than 100 ms greater than the current window center.

Using the <response> element, it is possible to customize various aspects of the window adjustment procedure, including [conditions for incrementing the window center](#) (moving it back), [conditions for decrementing the window center](#) (moving it forward), the [increment](#) and [decrement](#) amounts, the [minimum](#) and [maximum](#) window center values, and whether to base the adjustment algorithm on [mean or median latency](#).

Response Window for Blocks

The response window procedure can be specified at the block level, as in the following:

```
<block myblock>
/ responsemode = window(100, 100, windowstim)
</block>
```

In this example, adjustment of the window center for *myblock* is independent of the subject's performance on different blocks defined in the script. Each adjustment to the response window of *myblock* affects only subsequent runs of *myblock* and is unaffected by performance on blocks with other names.

Response Window for Expt

The response window procedure can be specified at the expt level, as in the following:

```
<expt>
/ responsemode = window(100, 100, windowstim)
</expt>
```

In this case, the scope of the window center is the experiment. All blocks within the experiment will share the experiment's window center, with the exception of blocks that have the response window explicitly defined for themselves (see above). Each block that uses the shared experiment window center may adjust that center based on the subject's performance, and subsequent blocks in the experiment will use the adjusted window center.

Response Window for the Response Element

The response window procedure can also be specified within a response element, as in the following:

```
<response myresponse>
/ mode = window
/ rwhitduration = 200
/ rwhitstimulus = hitstim
/ rwcenter = 400
/ rwwidth = 300
/ rwstimulus = windstim
</response>
```

In this case, the *myresponse* element has its own response window center, and all block or expt elements that specify `"/response = myresponse"` will share that window center.

This window center may be adjusted based on performance of any block that uses *myresponse*, and the adjusted window center will be used by subsequent blocks that use *myresponse*.

How to Control Response Timing

By default, Inquisit measures response latency as the interval beginning at the onset of the last stimulus frame of the trial and ending when the subject issues a valid response. Responses made before the onset of the last stimulus frame are ignored.

The beginning of the response interval can be customized using the [responsetime](#) or [responseframe](#) attributes. [Responsetime](#) specifies how many milliseconds after the onset of the first frame the response interval should begin. [Responseframe](#) specifies how many frames after the first frame the response interval should begin.

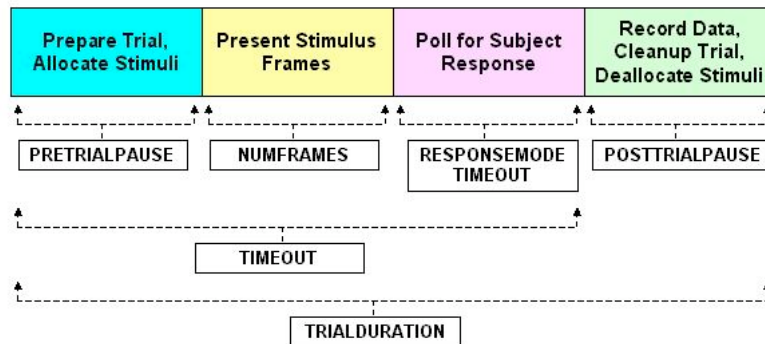
For example, in the following trial, the response interval begins when the first stimulus, "prime", is presented.

```
<trial mytrial>  
/ stimulusframes=[1=prime; 10=target]  
/ responseframe=1  
</trial>
```

How to Control Trial Duration and Inter-Trial Intervals

Inquisit provides a number of commands for controlling the timing of various segments of a trial. A trial can be thought of as a sequence of four segments, illustrated in the following diagram:

The Timeline of a Trial



Attributes

pretrialpause

Pauses for the specified duration at the beginning of a trial, prior to stimulus presentation. In addition to providing a general means of controlling inter-trial intervals, the PretrialPause is useful for experiments that present large numbers of memory intensive stimuli on a given trial. Depending on the size of the stimuli and the speed of the hardware, stimulus preparation may add notable lengths of time to the beginning of the trial. Furthermore, stimulus preparation time may vary from trial to trial, in which case varying durations may be added to the beginning of the trials. However, if a PretrialPause interval is specified, Inquisit uses this time to prepare the stimulus presentation sequence. By specifying a PretrialPause duration long enough for stimulus preparation to complete, the experimenter can impose a constant and predictable duration at the beginning of each trial.

numframes

Specifies the number of stimulus presentations frames. A frame corresponds to a single vertical retrace interval of a CRT monitor.

response

By setting this attribute to a timeout procedure (e.g., /response = timeout(1000)), it specifies the maximum duration for Inquisit to wait for the subject to respond. If no response occurs during within this duration, Inquisit finishes up the trial, waits for the posttrialpause to complete, and moves onto the next.

timeout

Specifies the maximum duration of a trial, from the very beginning of the trial to the end, not including the posttrialpause.

posttrialpause

Pauses for the specified duration at the end of each trial after the subject has responded. In addition to providing a general means of controlling inter-trial intervals, the PosttrialPause is useful for experiments that present large numbers of memory intensive stimuli on a given trial. Depending on the size of the stimuli and the speed of the hardware, the process of cleaning up a stimulus presentation sequence (i.e., removing stimuli from memory) may add notable lengths of time to the end of the trial. Furthermore, stimulus cleanup time may vary from trial to trial, in which case varying durations may be added to the ends of the trials. However, if a PosttrialPause interval is specified, Inquisit uses this time to cleanup the stimulus presentation sequence. By specifying a PosttrialPause duration long enough for stimulus cleanup to complete, the experimenter can impose a constant and predictable duration at the end of each trial.

trialduration

Specifies the absolute duration of a trial, from beginning to end, including the posttrialpause. If the subject responds quickly, the posttrialpause interval is lengthened to fill out the remaining time in the duration. If the subject does not respond before the duration, the trial is terminated and the next trial begins.

How to Setup and Use Setup Speech Recognition

Inquisit leverages the Microsoft Speech Recognition Engine to allow measurement of both latency and accuracy of spoken responses. To use Inquisit's speech recognition capabilities, you must install version 4.0 of the Microsoft Speech Recognition Engine and and Speech API. If you have not already installed these components, you may do so here. Simply click on each link below and when prompted choose the option to "*Run this program from its current location*".

1. [Install the Microsoft Speech API](#)
2. [Install the Microsoft Speech Recognition Engine](#)

The speech recognition functionality also requires that you have a sound card and a microphone. Almost any sound card will work for speech recognition and text-to-speech, including Sound Blaster™, Media Vision™, ESS Technology, cards that are compatible with the Microsoft® Windows Sound System, and the audio hardware built into multimedia computers. The quality of the microphone is a large determinant of speech recognition accuracy. Use a close-talk or headset microphone that is held close to the mouth or a medium-distance microphone that rests on the computer 30 to 60 centimeters away from the speaker. A headset microphone is needed for noisy environments. Speech recognition works best with close-talk microphones.

Before speech recognition will work, you must run the Microphone Wizard to adjust microphone input to appropriate levels. To run the Microphone Wizard, go to the *Tools* menu in Inquisit, select *Speech Recognition*, and then select *Microphone Wizard*.

How to Present TTL Signals Through the Parallel Port

Inquisit can trigger external devices such as EEG's and pidgeon feeders by sending these devices TTL (Transistor-Transistor Logic) signals through the computer's LPT (parallel) port. Inquisit allows precise control over the the duration and state of the signals, as well as synchronization of signals with visual and audio stimuli presented on a trial.

A TTL signal is a simple 8 bit value that is physically represented as the sequence of high and low voltage states in pins 2 through 9 of the parallel port at a given point in time.

In Inquisit's scripting language, parallel port signals are defined and presented much the same way as visual and audio stimuli. as the signal is represented as an 8 character sequence of 0's and 1's, with 0 corresponding to low and 1 corresponding to high. Port signals are defined within Inquisit using the `port` element, which can contain 1 or more items that define a specific 8-bit sequence. For example, the following port stimulus consists of a single 8 bit sequence "10101010":

```
<port mysignal>  
/ port = lpt1  
/ subport = data  
/ items = ("10101010")  
</port>
```

The **port** attribute specifies which of the two parallel ports to use, LPT1 or LPT2.

The **subport** attribute specifies whether to use the Data or Control register of the parallel port.

The **items** attribute specifies a sequene of 8 bits to send to the port.

This port signal can then be presented along with other trial stimuli:

```
<trial mytrial>
...
/stimulusframes = [1 = mypicture, mysignal]

< BR
> ...
</trial>
```

The standard parallel port consists of three registers or 'subports' commonly referred to as Data, Status and Control registers. The Data and Control registers are capable of sending output signals. Inquisit supports writing to either the Data and Control registers by defining the **subport** attribute of the **port** element. The Status register, on the other hand, supports input signals. Inquisit supports reading from Status register (see ?? for details).

TTL signals are specified as a sequence of 8 bits (i.e., a byte of information). The following charts show the mappings between each binary digit specified in the item (e.g., "01001001"), the bit, and their respective DB25 pins for signals send to the Data and Control registers, respectively:

Data Register		
Item	Bit	DB25 Pin
1	7	9
2	6	8
3	5	7
4	4	6
5	3	5
6	2	4
7	1	3
8	0	2
Control Register		
1	7	not used
2	6	not used
3	5	not used
4	4	not used
5	3	17
6	2	16 (logic reversed)
7	1	14
8	0	1

Inquisit's parallel port triggering is a generic mechanism for sending any 8-bit TTL signal to any external device capable of receiving. The duration and content of the signals sent by an experiment depends upon the specific device that is listening for those signals. Consult the documentation for your device to understand what kind of signals it expects.

Inquisit includes a [parallel port monitoring tool](#) that allows ad hoc sending of TTL signals to the Data and Control registers of either of two parallel ports (LPT1 or LPT2).

Inquisit can also read signals from the parallel port sent from other devices using the [pretrialsignal](#) and [posttrialsignal](#) commands.

How to Use the Parallel Port Monitor Tool

The Parallel Port Monitor tool can be launched by selecting the "Parallel Port Monitor..." command from Inquisit's "Tool" menu.

The Parallel Port Monitor tool allows you to send TTL signals through the parallel port of your choice, as well as monitor incoming TTL signals sent from external devices. The parallel port can be useful for testing what kinds of TTL signals are sent or recognized by external devices such as EEGs amplifiers.

Inquisit recognizes TTL input to the Status register. When the "Receive" button is pressed, the Parallel Port Monitor tool displays both the byte value of the current input TTL signal as well as the high/low status of each individual pin. A checked box indicates high; unchecked indicates low.

Inquisit can send TTL input to the Data or Control registers. To send a signal, check the box for pin that should be high, then press the "Send" button. The corresponding pins will be set high or low and will remain in that state until another signal is sent.

Configuring a Cedrus RB-series Response Box to Work with Inquisit

Inquisit supports obtaining and measuring responses from Cedrus RB-series response boxes. This article describes how to configure your response box and port for use with Inquisit.

Inquisit RB-x20 Response Boxes (RB-420, RB-520, RB-620, and RB-820)

The RB-x20 devices are the second generation of response boxes from Cedrus. These response boxes support only the "RB Series" communication mode.

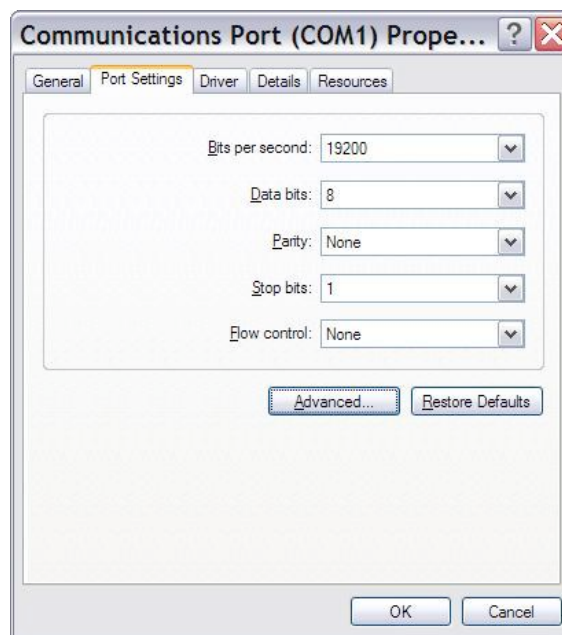
Setting the Dipswitches

The dipswitches on the back of the response box control, among other things, the baudrate that the device uses to signal the serial port. When the box is shipped, all four switches on the device are down by default, which corresponds to a baud rate of 19200 (see the image below). Although Inquisit supports any baud rate, we'll just use the default setting of 19200 to keep things simple. If the switches are not all in the down position, unplug your device from the computer, flip them all down, and then plug it back in.



Configuring the Serial Port

Once all four dipswitches have been set to the down position, the serial port must be configured with the compatible settings. To configure the port, use the Windows Device Manager applet (on Windows XP, open the Control Panel, then the System applet, click on the Hardware tab, and click the Device Manager button). Expand the Ports node on the tree, right click on the COM port that your response box is plugged into, and select the Properties command. This will open the window pictured below, which allows you to configure the port. For the RB-x20 devices with the dipswitches in the down position, the port should be set to the following:



If you haven't already, plug your device into the serial port of your computer. If your computer has multiple serial ports (COM1, COM2, etc.), make sure that you connect the device to the same port that you configured in the previous step.

Testing with Inquisit

To test Inquisit with your response, download and run the RB Mode Sample script from [Inquisit Samples Page](#). You can use this script to determine whether Inquisit is able to detect key presses, and what value is associated with each of the keys. This script works for both RB-x20 and RB-x30 devices.

Inquisit RB-x30 Response Boxes (RB-430, RB-530, RB-630, and RB-830)

The RB-x30 devices are the third and, as of the time this article was written, latest generation of response boxes from Cedrus. These response boxes support two modes of communication, the classic "RB Series" mode as well as the new XID mode. Either mode can be used with Inquisit. In XID mode, Inquisit uses the response box's built-in timer to determine response latencies. In theory, the response box's timer should be more reliable than the computer's CPU given that Windows allows other applications to take control of the CPU, although in practice, our testing has found little if any difference between response latencies measured in RB and XID mode.

The first step in using the response device is to plug it into a USB port on your computer and install the device driver (which you can download from Cedrus' web site).

XID Mode

XID Mode: Setting the Dipswitches

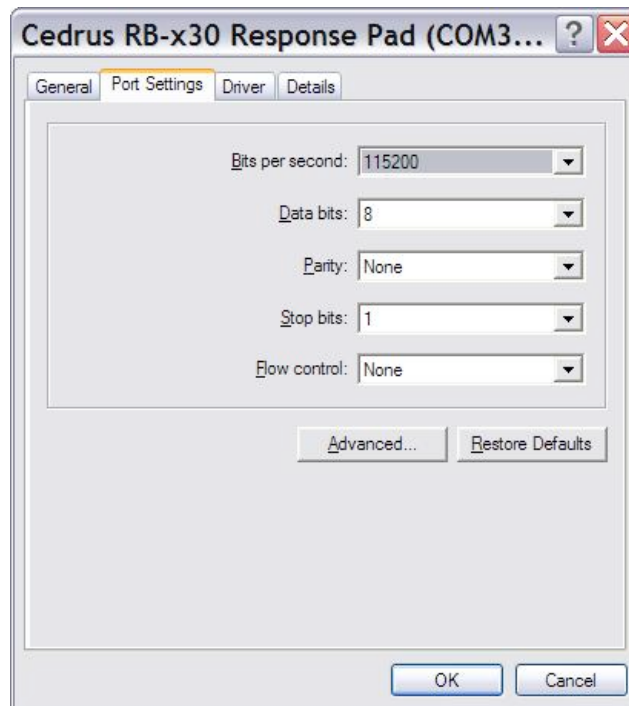
The dipswitches on the back of the response box control the response mode and baudrate that the device uses to signal the serial port. When the response box is shipped, all four switches on the device are down by default, which sets the device to XID mode at a baud rate of 115,200. See below for an illustration. Although Inquisit can use any baud rates supported by the device, we'll just use with the default setting of 115,200 for the sake of simplicity. If the dipswitches are in different positions, unplug the response box, return them all to the down position, then plug it back in. Be warned, you must disconnect your response box from the computer and then reconnect it for dipswitch changes to take effect.



XID Mode: Configuring the Serial Port

Once all four dipswitches have been set to the down position, we must make sure the serial port is configured to with the correct settings. To check the settings, use the Windows Device Manager applet (on Windows XP, open the Control Panel, then the System applet, click on the Hardware tab, and click the Device Manager button).

Expand the Ports node on the tree, right click on the port labeled "Cedrus RB-x30 Response Pad", and select the Properties command. (If you don't see this port in the tree, you need to install the device driver supplied by Cedrus.) This will open the window pictured below. Make sure the settings are as indicated below. (These are the defaults, so unless you've changed them, they should be.)



XID Mode: Testing with Inquisit

To test Inquisit with your response box in XID mode, download and run the XID Mode Sample script from [Inquisit Samples Page](#). You can use this script to determine whether Inquisit is able to detect key presses, and what value is associated with each of the keys.

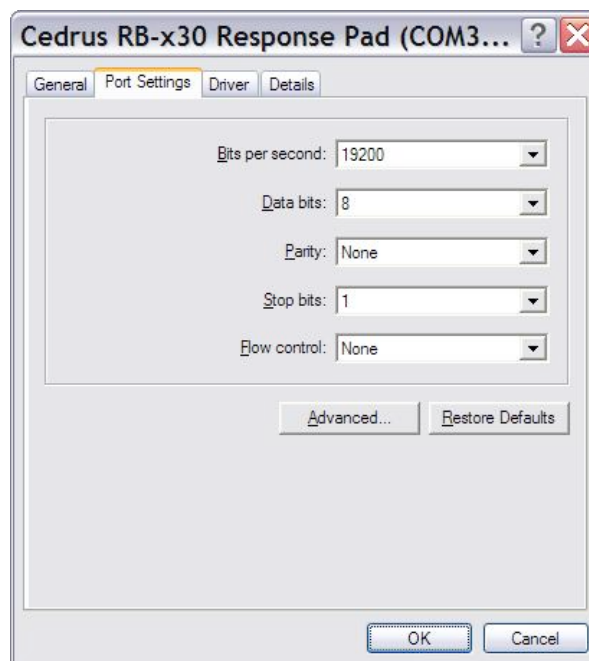
RB Mode

RB Mode: Setting the Dipswitches

The dipswitches on the back of the response box control the mode and baudrate that the device uses to signal the serial port. Unplug the response box from the computer, and set dipswitches 1 and 2 to the *up* position, and 3 and 4 to the *down* position. Be warned, you must disconnect your response box from the computer and then reconnect it for the changes to take effect. This will configure the device to use RB mode at a baud rate of 19200. Although Inquisit can use any baud rates supported by the device, 19200 is a perfectly respectable baud rate so we'll just stick with it to keep things simple.

RB Mode: Configuring the Serial Port

Once the dipswitches have been set to the proper position, the serial port must be configured with the appropriate settings. To check the settings, use the Windows Device Manager applet (on Windows XP, open the Control Panel, then the System applet, click on the Hardware tab, and click the Device Manager button). Expand the Ports node on the tree, right click on the port labeled "Cedrus RB-x30 Response Pad", and select the Properties command. (If you don't see this port in the tree, you need to install the device driver supplied by Cedrus.) This will open the window pictured below. Make sure the settings are as indicated below:



RB Mode: Testing with Inquisit

To test Inquisit with your response box in RB mode, download and run the RB Mode Sample script from [Inquisit Samples Page](#). You can use this script to determine whether Inquisit is able to detect key presses, and what value is associated with each of the keys. This script works for both RB-x20 and RB-x30 devices.

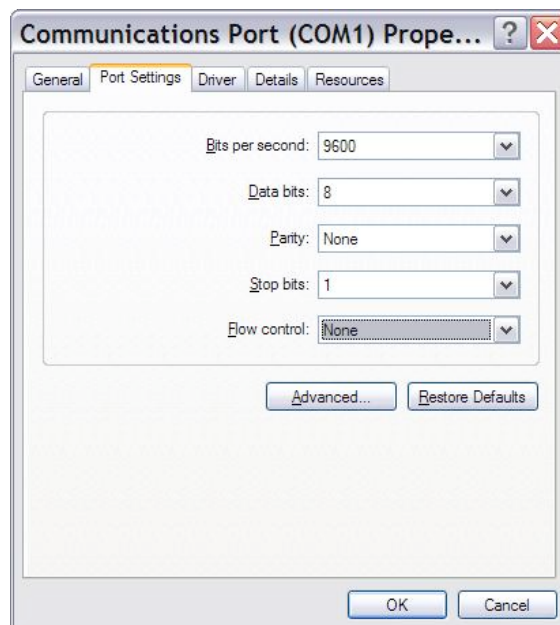
Record Responses from a Serial Response Box

Inquisit supports obtaining and measuring responses from serial port devices such as serial response boxes and voice key microphones (Inquisit also supports voice key and speech recognition using any PC microphone). Inquisit has generic serial port capabilities that can be used to interact with many devices. In addition, Inquisit also has special capabilities for interacting with Cedrus' XID response boxes, which are described in detail below.

Configuring the serial port

Most devices require the serial port to be configured with specific settings in order to function properly. Settings include the port's baud rate, the number of bits in each chunk of data, parity, and flow control. Consult the manual for your device for the appropriate COM port settings.

To configure the port, use the Windows Device Manager applet (on Windows XP, open the Control Panel, then the System applet, click on the Hardware tab, and click the Device Manager button). Expand the Ports node on the tree, right click on the COM port you wish to configure, and select the Properties command. This will open the following Window, which allows you to configure the port. If you are using an RB x30 series device, the settings should be as follows:



Note that the RB x30 series of response pads have a set of dip switches on the back that allow you to configure the device to use a 9600 or 19200 baud rate. Be sure to set the "bits per second" option in the above window to whichever of these two values the dip switches are configured to use.

Inquisit commands for serial port input

The Inquisit commands for using a serial port device are very similar to those used for other response devices such as a keyboard, mouse, or joystick. The inputdevice command should be set to "com1", "com2", "com3", ... depending upon which port number the device is plugged into on your PC. The inputdevice can both be set globally for all trials in an experiment by using the <defaults> element:

```
<defaults>  
/ inputdevice = com1  
</defaults>
```

The inputdevice can also be set on a trial by trial basis as follows (trial settings override the default settings):

```
<trial mytrial>  
/ inputdevice = com2  
/ validresponse = (128, 8)  
/ correctresponse = (8)  
</trial>
```

Next, define the valid and correct responses for that trial. Inquisit treats each byte of data sent by the device as a separate response whose values range from 0 to 255. Correct and valid responses must therefore be an integer within this range. The exact value sent by a particular device, or by a given button on a device, is determined by the device itself. To determine the value corresponding to a particular response on a serial port device, consult the documentation for that device, or run a series of trials with "responsemode=anyresponse". Under this mode, Inquisit will treat any response from the device as valid and will record the value for each response in the data file. You can then consult the data file to determine the value for each response.

Using Cedrus XID-compatible response boxes with Inquisit

Cedrus' new line of RB and Lumina response boxes include a built in timer that can be used to measure response latencies. The response pad measures the participant's reaction using it's own hardware and then reports a time stamped response to the computer. This capability is useful when running computationally intensive trials such as video presentation that could affect the computer's ability to provide accurate timing.

Taking advantage of an XID response box's on board timer is easy. There is no port configuration required. Just set the inputdevice command to "xid1", "xid2", "xid2", ... depending on which serial port number the device is plugged into. That's all there is to it. Inquisit will now record response latencies using the device's timer rather than that of the computer.

Using Current Design, Inc. fORP response devices with Inquisit

Inquisit is also compatible with fORP response devices by Current Design, Inc. These devices plug in through a USB port and can be configured to register as a keyboard or joystick device. Inquisit can thus interact with them as if they were a regular keyboard or joystick (i.e., by setting inputdevice equal to "keyboard" or "joystick").

How to Run an Inquisit 3 Experiment on the Web

Inquisit 3 Web Edition allows you to launch your experiments directly from a web page. If you have purchased a web license, you have the option of launching experiments from your own web site or from the millisecond.com web site. In either case, data are saved by default to the millisecond.com data service where you can login and download the data files.

If you haven't yet purchased a web license, you can still evaluate Inquisit 3 Web Edition by setting up an experiment on your own web server. When evaluating Inquisit, you can launch and run scripts as normal, but the data will not be saved.

[Click here](#) for more information on registering Inquisit 3 Desktop Edition. [Click here](#) for more information on registering Inquisit 3 Web Edition.

Publishing Inquisit scripts on millisecond.com

Hosting your scripts on millisecond.com is the easiest option for those without experience creating and administering web sites. For those with basic web development skills, this option also includes some support for customizing the launch web page and subject number assignment method. To publish a script on millisecond.com:

1. Write and test your Inquisit script using the Inquisit 3 Desktop Edition editor and tools.
2. Open your web browser and navigate to the millisecond.com web site.
3. Select "My Account" from the menu and click the "Register Inquisit Web Scripts" menu item. If you are not already logged into the site, you will be prompted for your user name and password.
4. Under the "Register Web Scripts" section, click the "Register New Script" link. This will launch the Inquisit Web Script Wizard
5. The first page of the wizard asks whether you wish to host the experiment on millisecond.com or on your own web server. Select the millisecond.com option. Then click the "Browse..." button and select your script file from your local computer. Click next once you have specified the script file.
6. On the next page you can upload additional files used by the script such as pictures and video.
7. Next, select whether you wish to use Inquisit's automatically generated launch page or your own custom web page. The subsequent steps in the wizard allow you specify the title, instructions, and how subject id numbers should be generated and assigned to subjects.
8. When you are done, click the Finish button. That's it, your experiment is now online. You can browse to the launch page using the following url:

[http://research.millisecond.com/\[username\]/\[scriptfilename\].web](http://research.millisecond.com/[username]/[scriptfilename].web)

where [username] is your user id and [scriptfilename] is the original filename of your script.

9. Click the "Start" link to launch your experiment.

Publishing Inquisit scripts on your own web server

Hosting experiments on your own server is an easy if you have access to a web server. To deploy an Inquisit experiment to your web server, follow these steps:

1. Write and test your Inquisit script using the Inquisit 3 Desktop Edition editor and tools, or download a script from the [Inquisit Task Library](#) .
2. Navigate to your web scripts page at <http://www.millisecond.com/myaccount/webscripts.aspx>.
3. If the status of your web license is "pending", start your web license by clicking the "Start Now" link.
4. Click the "Register New Script" link to launch the registration wizard and follow the steps in the wizard.
5. On the first page of the wizard, select the option to host the experiment on your own server, and enter the full url to the script file on your server.

6. Continue through the wizard specifying the options you'd like for the launch page.
7. On the final Summary page of the wizard, click the "Download Launch Page" button and save the html page to your computer. Then click the Finish button.
8. Upload your script file and the launch page created above to the location on your web server that you specified when registering the script. If your script uses picture or other media files, be sure to upload those as well.
9. Direct participants to the launch web page to start the experiment.

How to Combine Multiple Data Files into a Single File

If you are saving [unencrypted](#) to a local folder, Inquisit saves all data to a single file. However, if you [encrypt](#) the data or save it to a [remote server](#), Inquisit creates separate data files for each run of the script. In either case, you will likely need to combine data files from multiple machines or multiple subjects into a single file for analysis.

You can combine multiple data files into a single file by selecting the "Open" command on Inquisit's file menu and multi-selecting all of the files you wish to combine. Inquisit will open all of the selected files, appending them together. You can then select the "Save As" command on Inquisit's File menu to save the combined data into a single file.

How to do Conditional Branching with Inquisit

Inquisit provides several attributes that enable an experimental procedure to dynamically change or adapt based on the subject's performance.

Branch Attribute

The most powerful attribute for conditional logic is the [branch](#) attribute. The branch attribute can be defined at the level of the [trial](#) (including specialized trials such as [likert](#), [openended](#), and [namingtrial](#)) or [block](#). The attribute allows you to specify which trials or blocks to run next based on performance. Branching is useful in a variety of circumstances:

- Repeating a task until the subject makes ten correct responses in a row.
- Running one of two tasks depending upon the median latency on a previous task.
- Running a minimum of 10 and a maximum of 50 trials in a block and moving onto the next block if the average response latency rises above 1000 milliseconds.

The syntax of the attribute is as follows:

```
/ branch = [if <boolean expression> then nextevent]
```

where <boolean expression> is an [expression](#) that evaluates to true or false, and nextevent is the trial or block that should be run if the boolean expression is true. To specify that no branching should occur if a particular condition is true, "0" can be specified as the next event.

Multiple branches may be defined for a given element. If the conditions of multiple branches are true, then the first branch in the list wins.

Skip Attribute

Another useful attribute for conditional logic is the [skip](#) attribute. Like the branch attribute, skip can be defined at the level of . The attribute allows you to specify conditions under which which the [trial](#) or [block](#) should be skipped. The syntax of the attribute is as follows:

```
/ skip = [<boolean expression>]
```

where <boolean expression> is an expression that evaluates to true or false. Multiple skip conditions may be defined for a given element. If any of the skip conditions of are true, this event is skipped. Otherwise, it runs as normal.

Responsetrial Attribute

The [responsetrial](#) attribute provides a convenient way to chain together trials based on which response the subject made. For example:

- Following up incorrect responses with a study trial.
- Creating a questionnaire that includes follow up questions only if a particular response is given.

Note that with Inquisit 3, the branch attribute includes all of the functionality of the responsetrial, and also includes functionality not supported by responsetrial.

Stop Attribute

The [stop](#) attribute aborts the remainder of the trials in a block if the subject's performance meets the specified condition. This is useful in a variety of situations:

- Aborting a task if the percentage of correct responses drops below a threshold.
- Aborting a task if an incorrect response is given.
- Aborting a task after a maximum number of trials has been run.

Running Sequences of Inquisit Scripts and Other Applications

A single experiment does not necessarily fit into a single Inquisit script. In many cases it may be necessary to collect data using a combination of different Inquisit scripts, or even a combination of different data collection programs. Ideally, the different scripts and programs would run as a single, seamless sequence of tasks requiring no manual intervention from the researcher. This article discusses approaches to achieve this.

Using the Inquisit Batch Element

The Inquisit language provides a simple facility for stringing together multiple scripts into a single data collection session, the `batch` element. The batch element allows you to specify a list of script files to run in order.

```
<batch>  
/ file="snowboard.exp"  
/ file="ski.exp"  
/ file="snowshoe.exp"  
</batch>
```

The batch element must be defined in its own separate file. To run the batch element, simply open the script containing the batch element definition in Inquisit and run it as you would run any other script.

This is a very simple approach, but it is also somewhat limited. The batch provides no way to launch other programs besides Inquisit, and it does not allow you to counterbalance the order in which the scripts are run. For that, we can rely on Windows batch files.

Using Windows Batch Files

Most researchers assume that running a sequence of data collection programs together requires the ability of the data collection software to launch other software applications. The applications can thus be daisy-chained together, with each application launching the next application in the sequence. In fact, this capability isn't really necessary at all. Everything you need to run batches of applications in sequence is built directly into the Windows operating system.

Those of you who remember the days of Microsoft DOS are probably familiar with batch files. Batch files are simple text files, usually named with the `*.bat` file extension, that contain a series of shell commands. Shell commands can move, copy, and delete files and folders. They can search for files by name or that contain a particular string of text. They can format a hard drive. Most importantly for our purposes, however, they can launch applications.

Creating a batch file

Creating a batch file is simple:

1. Click the Windows Start button in the lower left corner, and select "My Computer".
2. Navigate to the C: hard drive by double clicking its icon.
3. Open the "File" menu, select the "New" command, and then select "Text Document".
4. Now, rename the text file you created to "millisecond.bat". Click "OK" when Windows asks you to confirm that the name change.
5. Right click the millisecond.bat file and select the "Edit" command. This will open the file in the Notepad text editor.

Now we're ready to enter in some commands. For the sake of example, let's assume that our experiment consists of two Inquisit scripts called "Snowboard.exp" and "Ski.exp". We'd like to run the "Snowboard.exp" script first, then have the subject play a game of Solitaire to serve as a distractor task before running the "Ski.exp" scrip. To launch the "Snowboard.exp" Inquisit script, we use the same syntax that we would use to launch the script from the Windows command line shell (hence, the name "shell commands"). The Inquisit.exe program accepts 2 command line parameters that specify the script to run and the subject number to use. Thus, the first command in the batch file is the following:

```
"C:\Program Files\Millisecond Software\Inquisit 3\Inquisit.exe" "C:\Myscripts\Snowboard.exp" 23
```

The first part of the command is the full path to the Inquisit executable file (yours may be different depending on where you installed Inquisit). The second part specifies the full path of the script to run. The third part is the subject number (more on this later). Now, on the second line of the file we'll add the command for launching the Windows Solitaire application, which changes our batch file to the following:

```
"C:\Program Files\Millisecond Software\Inquisit 3\Inquisit.exe" "C:\Myscripts\Snowboard.exp" 23  
"C:\Windows\System32\sol.exe"
```

Finally, we'll add a third line to the file to run the "Ski.exp" Inquisit script.

```
"C:\Program Files\Millisecond Software\Inquisit 3\Inquisit.exe" "C:\Myscripts\Snowboard.exp" 23  
"C:\Windows\System32\sol.exe"
```

```
"C:\Program Files\Millisecond Software\Inquisit 3\Inquisit.exe" "C:\Myscripts\Ski.exp" 23
```

Our batch file is almost complete. We can run the file either by double clicking the file with the mouse, or by opening the Windows command line, moving to our desktop directory, and typing "millisecond.bat". The batch file first runs "Snowboard.exp". When that script is complete, it opens the Windows Solitaire application. When Solitaire is closed, it runs the "Ski.exp" script in Inquisit. In both cases, it sets the subject number to "23".

Wait a minute, you may be thinking, won't this batch file set assign "23" to all of my subjects? The answer is yes. Obviously, that's not very useful as we'll need the ability to specify a unique subject number for each participant. Fortunately, this isn't difficult because Windows allows you to pass command line parameters to batch files as well. In our case, we'll want to pass the subject number to the batch file, and have it apply that number to both Inquisit scripts. We can do that by modifying the batch file as follows:

```
"C:\Program Files\Millisecond Software\Inquisit 3\Inquisit.exe" "C:\Myscripts\Snowboard.exp" %1
"C:\Windows\System32\sol.exe"
```

```
"C:\Program Files\Millisecond Software\Inquisit 3\Inquisit.exe" "C:\Myscripts\Ski.exp" %1
```

All we've done is replaced "23" with "%1". Windows will substitute the value of the first command line parameter passed to the batch file wherever it see's a "%1". (All occurrences of "%2" would be replaced by the second parameter, "%3" by the third parameter, and so forth).

We've now got a batch file that will run the three pieces of our experiment in sequence, but what if we want to counterbalance the order of our Inquisit scripts? No problem, we'll just add some additional logic to our batch file. Our batch file is going to need a second command line parameter that tells us the order to run the Inquisit's scripts and an "IF" statement to select the correct order.

```
IF "%2" == "A" (
```

```
"C:\Program Files\Millisecond Software\Inquisit 3\Inquisit.exe" "C:\Myscripts\Snowboard.exp" %1
"C:\Windows\System32\sol.exe"
```

```
"C:\Program Files\Millisecond Software\Inquisit 3\Inquisit.exe" "C:\Myscripts\Ski.exp" %1
)
```

```
IF "%2" == "B" (
```

```
"C:\Program Files\Millisecond Software\Inquisit 3\Inquisit.exe" "C:\Myscripts\Ski.exp" %1
"C:\Windows\System32\sol.exe"
```

```
"C:\Program Files\Millisecond Software\Inquisit 3\Inquisit.exe" "C:\Myscripts\Snowboard.exp" %1
)
```

Now our batch file takes two command line parameters, the first is the subject number and the second (either "A" or "B") indicates which order to run. If the second parameter is neither "A" or "B", neither condition is run.

Running the batch file

Our batch file is now finished, so let's take a look at how to run it. We can no longer double click the file to run it, because the file is expecting us to supply a subject number as a first parameter. Instead, we'll launch the batch file from the Windows command line shell. To open the command shell, do the following:

1. Click the Windows Start button in the lower left corner of the screen.
2. Select the "Run..." command.
3. Enter "cmd" in the text input box, and click the "OK" button. This will open a command line console window.
4. In the console window, type "cd c:\ " to navigate to the root folder of the C drive (if you are using a different drive, then switch the drive letter).
5. To launch your batch file, type "millisecond.bat 99 A". Note that 99 is the value of the the subject number and A specifies the first counterbalancing condition. To run the second counterbalancing condition, you would type "millisecond.bat 99 B"

We now have a batch file that allows us to run multiple Inquisit scripts in counterbalanced order, as well as running a another software application, "Solitaire". To the participant, these different pieces will seamlessly flow together. [Click here](#) for the completed batch file.

Batch files are a powerful tool that can be used to automate many different aspects of data collection, including automatically backing up data files to another location when a session is complete, preprocessing data, or even logging off of Windows after a script is complete to prevent tampering (using the Windows "logoff.exe" program).

How to Combine Multiple Scripts

Often a study will involve administering multiple measures to each participant. For example, a study might administer an IAT, a direct measure of attitude, and a demographic questionnaire, or it may require that 2 different IAT tasks be administered. Inquisit places no constraints on the number of measures that can included in a single script. However, for a number of reasons, it is often more convenient for each measure to be defined in its own script. For example, putting all of the measures in a single file may result in a large and unwieldy script. Also, in many cases each of the measures already exists as a separate script anyway.

Inquisit provides two mechanisms for combining different scripts into a single data collection session, the <batch> element and the <include> element.

The <batch> Element

The batch element provides a simple way of running a set of scripts in sequence. To use the batch element, create new empty script file that will contain only the batch element definition and no other commands:

```
<batch>  
/ file = "IAT1.exp"  
/ file = "IAT2.exp"  
/ file = "IAT3.exp"  
</batch>
```

As you can see from this example, the batch element is really just a list of the script files you wish to run. In the above example, three IAT scripts are listed, IAT1.exp, IAT2.exp, and IAT3.exp. As you might have guessed, the batch runs each of these scripts in the order they are listed. To run the batch, simply open the script file that contains the batch element definition in Inquisit, and select the Run command from the Experiment menu. You will be prompted for a subject id, after which each script is run in sequence. Data from each script are saved in separate files, and the same subject id is used for each file. To use the batch element with Inquisit Web Edition, simply register the script containing the batch element, and then upload the other scripts to the web server. With the web edition, the data are also saved in different files using the same subject id.

The batch file is very easy to use, but it does have some limitations. Most notably, there is currently no built in way to randomize the order in which the scripts are run (this is true as of version 3.0.2.0, but the feature is planned for a future release). If you are using the desktop version of Inquisit, you can create multiple batch files, each of which contains a different ordering, and then randomly assign participants to one of those batch files. With the web edition, this technique would require you to register multiple batch scripts, each of which would require a separate license. For web experiments requiring randomized ordering, the <include> element described below is likely a better option.

The <include> Element

The include element provides a convenient way to "copy and paste" elements from one script into another script without requiring you to go through the hassle of actually copying and pasting. It therefore can serve as a useful tool for combining measures defined in separate scripts, although typically the scripts will have to be modified somewhat in order to combine properly.

As an example, let's say we wish to run two IAT measures, randomly varying the order. We could arbitrarily choose either IAT script and add the include element to it. However, in order to make it easier to reuse this solution with other scripts (containing IATs or other measures), we'll start with a new empty file and then add our include definition to it:

```
<include>  
/ file = "IAT1.exp"  
/ file = "IAT2.exp"  
</include>
```

In the above example, we've included two different scripts, IAT1.exp and IAT2.exp. The order in which they are listed doesn't matter. Conceptually, we now have a single virtual script that contains all of the element definitions contained in IAT1.exp and IAT2.exp. If you try to run the script, however, you'll notice a whole bunch of errors. This is because both scripts use the same names for elements. They also both include definitions of global elements such as <data>, <defaults>, <values>, <variables>, and <expressions>, so Inquisit will report an error stating that these elements have been defined more than once.

The first step in resolving these errors is to put a single copy of the global elements in our include script, and remove these element definitions from both of the IAT scripts. The global elements to add to our include script and remove from our IAT scripts are as follows:

```
<defaults>
/ fontstyle = ("Arial", 3.5%)
/ screencolor = (0,0,0)
/ txbgcolor = (0,0,0)
/ txcolor = (255, 255, 255)
/ minimumversion = "3.0.0.0"
</defaults>
```

```
<data>
/ columns = [date, time, subject, blockcode, blocknum, trialcode, trialnum, response, correct,
latency, stimulusnumber, stimulusitem, expressions.da, expressions.db, expressions.d ]
</data>
```

```
<monkey>
/ latencydistribution = normal(500, 100)
/ percentcorrect = 90
</monkey>
```

```
<values>
/ sum1a = 0
/ sum2a = 0
/ sum1b = 0
/ sum2b = 0
/ n1a = 0
/ n2a = 0
/ n1b = 0
/ n2b = 0
/ ss1a = 0
/ ss2a = 0
/ ss1b = 0
```



```

/ss2b = 0
/magnitude = "unknown"
</values>

<expressions>
/m1a = values.sum1a
/values.n1a
/m2a = values.sum2a
/values.n2a
/m1b = values.sum1b
/values.n1b
/m2b = values.sum2b
/values.n2b
/sd1a = sqrt((values.ss1a - (values.n1a * (expressions.m1a * expressions.m1a))) / (values.n1a - 1))
/sd2a = sqrt((values.ss2a - (values.n2a * (expressions.m2a * expressions.m2a))) / (values.n2a - 1))
/sd1b = sqrt((values.ss1b - (values.n1b * (expressions.m1b * expressions.m1b))) / (values.n1b - 1))
/sd2b = sqrt((values.ss2b - (values.n2b * (expressions.m2b * expressions.m2b))) / (values.n2b - 1))
/sda = sqrt((((values.n1a - 1) * (expressions.sd1a * expressions.sd1a) + (values.n2a - 1) * (expressions.sd2a * expressions.sd2a)) + ((values.n1a + values.n2a) * ((expressions.m1a - expressions.m2a) * (expressions.m1a - expressions.m2a)) / 4)) / (values.n1a + values.n2a - 1))
/sdb = sqrt((((values.n1b - 1) * (expressions.sd1b * expressions.sd1b) + (values.n2b - 1) * (expressions.sd2b * expressions.sd2b)) + ((values.n1b + values.n2b) * ((expressions.m1b - expressions.m2b) * (expressions.m1b - expressions.m2b)) / 4)) / (values.n1b + values.n2b - 1))
/da = (m2a - m1a) / expressions.sda / db = (m2b - m1b) / expressions.sdb
/d = (expressions.da + expressions.db) / 2
/preferred = "unknown"
/notpreferred = "unknown"
</expressions>

```

The next step is to rename all of the blocks, trials, stimuli, and stimulus items in both IAT scripts so that they are all unique. In our case, we'll simply add an "iat1" to beginning of all of the names in the IAT1.exp script, and "iat2" to the beginning of all of the names in the IAT2.exp script. Thus, the blocks named "targetcompatiblepractice" becomes "iat1targetcompatiblepractice" and "iat2targetcompatiblepractice". The text stimulus named "instructions" becomes "iat1instructions" and "iat2instructions". The trial named "iat1summary" and "iat2summary". And so forth. Note that you will need to update the parts of each script that refer to these elements as well. For example, iat1summary trial contains the command / stimulustimes = [0=summary], which must be changed to / stimulustimes = [0=iat1summary].

The element responsible for running our IATs is the <expt>element, so we'll next need to remove these element definitions from the IAT scripts and rewrite them in our include script so that they run the blocks of each of our IATs. After we delete these elements from the IAT scripts, we'll add the following element definition to the include scripts.

```
<expt>
/ subjects = (1 of 4)
/ blocks = [1=iat1targetcompatiblepractice; 2=iat1attributepractice; 3=iat1compatibletest1;
4=iat1compatibletestinstructions; 5=iat1compatibletest2; 6=iat1targetincompatiblepractice;
7=iat1incompatibletest1; 8=iat1incompatibletestinstructions; 9=iat1incompatibletest2;
10=iat1summary; 11=iat2targetcompatiblepractice; 12=iat2attributepractice;
13=iat2compatibletest1; 14=iat2compatibletestinstructions; 15=iat2compatibletest2;
16=iat2targetincompatiblepractice; 17=iat2incompatibletest1;
18=iat2incompatibletestinstructions; 19=iat2incompatibletest2; 20=iat2summary ]
</expt>
```

The above experiment runs IAT1 first and IAT2 second. In both cases, the compatible pairings are applied before the incompatible pairings.

The next expt element reverses the order of the IATs.

```
<expt>
/ subjects = (2 of 4)
/ blocks = [1=iat2targetcompatiblepractice; 2=iat2attributepractice; 3=iat2compatibletest1;
4=iat2compatibletestinstructions; 5=iat2compatibletest2; 6=iat2targetincompatiblepractice;
7=iat2incompatibletest1; 8=iat2incompatibletestinstructions; 9=iat2incompatibletest2;
10=iat2summary; 11=iat1targetcompatiblepractice; 12=iat1attributepractice;
13=iat1compatibletest1; 14=iat1compatibletestinstructions; 15=iat1compatibletest2;
16=iat1targetincompatiblepractice; 17=iat1incompatibletest1;
18=iat1incompatibletestinstructions; 19=iat1incompatibletest2; 20=iat1summary ]
</expt>
```

The next expt element runs IAT1 first and IAT2 second, but incompatible pairings are used first, and compatible pairings second.

```
<expt>
/ subjects = (3 of 4)
```

```

/ blocks = [1=iat1targetincompatiblepractice; 2=iat1attributepractice; 3=iat1incompatibletest1;
4=iat1incompatibletestinstructions; 5=iat1incompatibletest2; 6=iat1targetcompatiblepractice;
7=iat1compatibletest1; 8=iat1compatibletestinstructions; 9=iat1compatibletest2;
10=iat1summary; 11=iat2targetincompatiblepractice; 12=iat2attributepractice;
13=iat2incompatibletest1; 14=iat2incompatibletestinstructions; 15=iat2incompatibletest2;
16=iat2targetcompatiblepractice; 17=iat2compatibletest1; 18=iat2compatibletestinstructions;
19=iat2compatibletest2; 20=iat2summary ]
</expt>

```

Finally, the last expt element runs IAT2 first and IAT1 second, with incompatible pairings first, and compatible pairings second.

```

<expt>
/ subjects = (4 of 4)
/ blocks = [1=iat2targetincompatiblepractice; 2=iat2attributepractice; 3=iat2incompatibletest1;
4=iat2incompatibletestinstructions; 5=iat2incompatibletest2; 6=iat2targetcompatiblepractice;
7=iat2compatibletest1; 8=iat2compatibletestinstructions; 9=iat2compatibletest2;
10=iat2summary; 11=iat1targetincompatiblepractice; 12=iat1attributepractice;
13=iat1incompatibletest1; 14=iat1incompatibletestinstructions; 15=iat1incompatibletest2;
16=iat1targetcompatiblepractice; 17=iat1compatibletest1; 18=iat1compatibletestinstructions;
19=iat1compatibletest2; 20=iat1summary ]
</expt>

```

Note the /subjects command in each <expt> element determines which conditions a participant is assigned to based on the subject number. Subjects 1, 5, 9, 13, etc are assigned to the first expt element, 2, 6, 10, 14, etc. to the second, 3, 7, 11, 15, etc. to the third, and 4, 8, 12, 16, etc. are assigned to the fourth expt element.

To run the script with Inquisit Desktop Edition, simply open the script containing the include element and select the Run command from the Experiment menu. To run it with Inquisit Web Edition, register the script with the include element, and upload the other two scripts to the server.

Using an ASL Eye Tracker with Inquisit

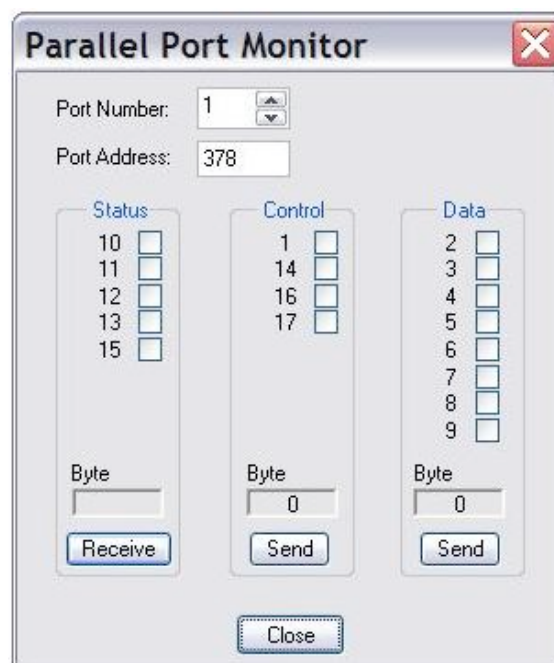
Versions 3.0.4.0 and later of Inquisit include a module that enables X-DAT markers to be sent from Inquisit to the eye tracker, and real time eye position data to be retrieved from the eye tracker for use in Inquisit experiments.

Sending Markers from Inquisit to an ASL Eye Tracker

Inquisit can send XDAT markers to the Eye Tracker in order to notify it of events in an experiment such as the onset of a stimulus or the moment at which a participant responds. The Eye Tracker then records these values into the data stream so that they can be used to analyze the tracking data. To send XDAT markers, follow these steps:

1. Using the parallel cable labeled "XDAT" supplied by ASL, connect the parallel port of your Inquisit computer to the XDAT port on the back of Eye Tracker Control Unit.
2. Start the Eye Tracker.
3. To test your connection, start Inquisit, select the Tools menu, and select the "Parallel Port Monitor" command. Using the port monitor tool, select the port that the XDAT cable is plugged into, check all of the boxes in the Data section (pins 2-9), and click the Send Button. The XDAT value reported by the Eye Tracker software should change to 255.
4. Open the sample script for "Sending Markers to ASL Eye Tracker" (this sample can be downloaded from <http://www.millisecond.com/download/samples/>). Run the sample by clicking on the "Experiment" menu and selecting the "Run" command. The script presents a series of pictures and sends a unique marker value at the onset of each one.

Figure 1: Inquisit's Parallel Port Monitor



Sending Data from an ASL Eye Tracker to Inquisit

Inquisit allows a response to be defined as when the participant's gaze falls within one or more predefined regions on the screen, such as the areas occupied by pictures, text, videos, or arbitrary shapes. Both the region that the participant looks at and the latency of their gaze can be recorded by Inquisit, reported back to the participant, and used to direct the branching and flow of the experiment.

The sample entitled "Retrieving Point of Gaze from ASL Eye Tracker" demonstrates how to use gaze points in an Inquisit experiment. To run the sample, download it from the Inquisit samples web page (<http://www.millisecond.com/download/samples/>) and follow these steps:

1. Set up and test serial connection between Eye Tracker Serial Out port and the Inquisit computer as described in ASL SDK Appendix: Testing Serial Out Connection.
2. Open the sample experiment in the editor.
3. The sample script assumes that the serial out cable is plugged into the COM1, and the XDAT cable is plugged into LPT1. It also assumes that head tracking is off. If you are using different port numbers, you can change those values in the script by locating the `<eyetracker>` element in the script (see figure 2 below) and changing the numbers in the following two lines:
 `/ comportnumber = "1"`
 `/ lptportnumber = "1"`

Figure 2: Eyetracker Element Example

```
<eyetracker>  
/ file = "ASLEyeTracker.dll"  
/ comportnumber = "1"  
/ eyeheadintegration = "false"  
/ lptportnumber = "1"  
</eyetracker>
```

4. Run the experiment by clicking the "Experiment" menu and selecting the "Run" command.
5. Inquisit will present a 9-point calibration bitmap on the screen. While this screen is displayed on the Inquisit computer, click the Set Target Points command on the Eye Tracker computer and set each of the target points so that the eye tracker can accurately map pixels on the Inquisit computer's to Eye Tracker units. Press the space to bar to continue.

6. Next, Inquisit will begin the process of calibrating the eye tracker to the participant's eye. To begin eye calibration, select the "Standard Calibration" command on the Eye Tracker computer, and Inquisit will present the current calibration point. As the participant is looking at the point, click the "Save Current Point" button and the next point will appear on the presentation screen. When you are finished calibrating, press the space bar on the Inquisit computer.
7. Inquisit will then present a single dot in the upper left corner of the screen. Instruct the participant to look at that point and press the space bar.
8. Inquisit will then present a single dot in the lower right corner of the screen. Instruct the participant to press the space bar as they look at the dot.
9. Inquisit will then present instructions for running the demo. The demo shows how to display a cursor on the screen at the current gaze point and report the current x and y pixel coordinates.

See the Inquisit documentation for more information on configuring the Eye Tracker and using gaze points in Inquisit.

Sending Markers from Inquisit 2.0 to Eye Tracker

Although Inquisit 2.0 does not support the ASL Eye Tracker module, it is nevertheless possible to send XDAT markers to the Eye Tracker using Inquisit's generic parallel port signaling capabilities. To send XDAT markers, follow these steps:

1. Using the parallel cable labeled "XDAT" supplied by ASL, connect the parallel port of your Inquisit computer to the XDAT port on the back of Eye Tracker Control Unit.
2. Start the Eye Tracker.
3. To test your connection, start Inquisit, select the Tools menu, and select the "Parallel Port Monitor" command. Using the port monitor tool, select the port that the XDAT cable is plugged into, check all of the boxes in the Data section (pins 2-9), and click the Send Button. The XDAT value reported by the Eye Tracker software should change to 255.
4. Define your port signals using Inquisit's `<port>` element. The `<port>` element is just another type of stimulus, similar to `<picture>`, `<text>`, `<video>`, and `<sound>`. As such, port stimuli can be presented using the same syntax that other stimuli can be presented.

How to Interoperate Inquisit Web Edition with Online Survey Packages

For many research projects, it is desirable to use Inquisit's reaction time capabilities in conjunction with online survey packages such as Survey Monkey, Unipark, Qualtrics, etc. This scenario is quite common among Inquisit users, and we've put some features in place to make the transition back and forth between Inquisit and other packages as smooth as possible.

Integrating surveys and Inquisit measures into a seamless and coherent user experience for participants is primarily accomplished by automatically redirecting participants back and forth between the different programs. Most survey packages allow you to forward participants to another web site after they've completed part or all of the survey. You need only specify the Inquisit launch page as your forwarding url, and your participants will be automatically redirected to the Inquisit portion of the study when the survey is complete.

Similarly, Inquisit allows you to specify a "Finish Page", where it will redirect participants after they've completed the Inquisit part of the experiment. Here you would specify the url to the survey page where you want participants to go next. Having done this, Inquisit will automatically send participants back to the survey once the reaction time task is complete.

Now the question is, how do you correlate the data from the survey with the Inquisit reaction time data? And, if you are sending participants back to the survey web site after the Inquisit part, how does the site know which participant this is? The answer is to create a unique subject id for each participant, and share it between Inquisit and the survey so that both record it in the data file.

The way to do this is to have participants start in the survey, all of which will generate some sort of unique id for participant. When the survey is finished, the survey package will forward them to the Inquisit launch page url, and it will append the participant id (and sometimes some other data) to the url as query parameters. For the example, the forwarding url to the Inquisit page might look like the following:

<http://research.millisecond.com/sniffles/myexperiment.web?subjectnumber=134&phase=3>

The url contains two parameters named "subjectnumber" and "phase", the values of which are dynamically set by the survey package for each participant (to 134 and 3 in this example). The "subjectnumber" parameter is a unique id, and "phase" is used by the survey package to determine which phase of the survey to run next when the participant returns. We want to make sure that a) Inquisit records this subject number in the data, and b) Inquisit forwards both parameters back to the survey web site when finished so the site knows which subject has arrived and can pick up where it left off.

To use the subjectnumber, just run through the webscript registraion wizard. When asked how to generate subject ids, select the "Query Parameter" option and specify the name of the parameter (in this case, the name is "subjectnumber"). That's it, Inquisit will now extract this subject number from the url and record it in the data file.

When the Inquisit is finished, it will automatically append all of these query parameters to the Finish Page url so that the values are passed back to the survey package. Continuing with the example above, if the finish page is:

<http://www.surveysrus.com/coolsurvey/part2.html>

Inquisit will append the parameters onto it so the actual forwarding url is

<http://www.surveysrus.com/coolsurvey/part2.html?subjectnumber=134&phase=3>

Now the survey package can extract these values and pick up where it left off with this participant.

You can specify the finish page using the webscript registration wizard as well. Importantly, you should just specify the base url here without the query parameters! Inquisit will dynamically append the parameters onto the base url for you.

Introducing Inquisit 3 Web Edition

Inquisit 3 Web Edition (IWE) extends the power and flexibility of the Inquisit 3 experiment engine to the web. With IWE, your experiments can be launched directly from a web page without having to manually install Inquisit on the client machine. Data gathered from the web can be saved back to a web site, ftp site, network share, or even an email address.

What are the benefits of Inquisit 3 Web Edition?

Unlimited client licenses. With IWE, Inquisit experiments can be run on an unlimited number of client machines. This makes IWE an ideal tool for large scale data collection in laboratories, class rooms, or over the Internet.

Easy web deployment. IWE allows you to launch your experiments directly from a web page. The IWE engine is packaged both as an ActiveX control and a Mozilla Plugin that is automatically downloaded by Internet Explorer and Mozilla browsers like Firefox and Netscape respectively. Starting an experiment with IWE is as easy as browsing to a web page.

Power, flexibility, and accurate timing. The IWE engine affords the same power, flexibility, and timing accuracy as the Inquisit desktop engine. How is that possible? Because the IWE engine *is* the desktop engine. Literally. We've just repackaged it in a way that makes it easy to deploy over the web.

Compatible with Inquisit 3 Desktop. Your Inquisit 3 scripts will run interchangeably between the desktop and web engines. There is no need to maintain and test multiple versions of your scripts.

Free Inquisit 3 Desktop license included. Each web license you purchase includes a free desktop license so that you can easily create and test your Inquisit scripts on your development machine before deploying them to the web.

How does the Inquisit 3 Web Edition work?

Running a web experiment requires that you have access to a web server. There are two simple steps for publishing your experiment. First, you must copy your Inquisit script to the web server so that the IWE can download it. Then, you publish the web page that will be used to launch the Inquisit experiment.

The launch page is just a standard html page where you can send your participants in order to start the experiment. For an example of a complete Inquisit launch page, navigate to <http://www.millisecond.com/web/launch.aspx> and use your browser's command for viewing the source. The launch page is formatted so that you can adapt it to run your own script with a few simple modifications. When the user browses to the launch page, the browser prompts the user to install the control, which is then downloaded from millisecond.com. Next, the page uses client-side javascript to call the RunScriptFile method on the IWE in order to launch the experiment. The RunScriptFile method signature is as follows:

```
Inquisit.RunScriptFile(scriptpath, userid, subjectnumber, monkey, encryptionpassword);
```

The *RunScriptFile* method takes four parameters:

scriptpath

The URL where IWE can download the script.

userid

The userid of the account that has specified the script as active.

subjectnumber

The subject number to use for this subject.

monkey

A boolean flag indicating whether Inquisit should run in monkey mode for the purpose of testing a script.

encryptionpassword

A string containing the password required to decrypt an encrypted script.

The following example runs a script on millisecond.com with subject number set to 1.

```
Inquisit.RunScriptFile("http://www.millisecond.com/myscript.exp", "myuserid", 1, false, "");
```

RunScriptFile will download and parse the script, and if there were no errors it creates a full screen window and runs the script in the same fashion as the Inquisit desktop engine. Using javascript, you can call this method when the page loads, when user clicks a button or types a key, or when any other event is fired by IE. This gives you great flexibility in determining how the experiment integrates with your web site.

How does licensing work with Inquisit 3 Web Edition?

Unlike Inquisit Desktop, IWE is not licensed based on the number of client computers that install and run Inquisit experiments. In fact, an IWE license entitles you to run Inquisit experiments on an unlimited number of client computers.

IWE licenses determine the number of experiments that can be run at a given point in time. An IWE license entitles you to run a single web experiment (as defined by a single Inquisit script). If you wish to run one web experiment at a time, you would only need a single IWE license. If you wish to collect data for five different experiments in parallel, you would need five IWE licenses.

IWE enforces the licensing policy at run time by checking whether the script it has been instructed to run is listed as an active script for the specified user account. If the script is active, the experiment runs as normal. If the script is not active, it can optionally be run, but no data will be collected. In order to collect data with IWE, you must therefore register the URLs of your active scripts using the [Inquisit Web Script Manager](#) web page. You may change your list of active scripts as often as you'd like. However, the number of active scripts you may specify is limited to the number of IWE licenses held by your account. Importantly, this means that IWE must be able to connect to www.millisecond.com from the client computer in order to collect data. *IWE can not be used to collect data on machines that are not connected to the Internet.*

What are the machine requirements for Inquisit 3 Web Edition?

The client must have a working Internet connection.

The client must be running Internet Explorer 4.0 or higher, or a browser based on Mozilla 1.0 or higher (e.g. Firefox 1.5 or Netscape 8.0).

The client must have DirectX 2.0 or higher.

The client must be running Windows 98, ME, 2000, XP, or 2003. IWE will not run on Windows 95 or NT4 (actually, it might, but we no longer test these versions of Windows so they are not officially supported). It will not run on any non-Windows operating systems. Sorry, no support for Mac or Linux.

Internet Explorer security options must be configured to enable ActiveX controls that are a) marked as "safe for scripting" and b) have a valid Authenticode signature. By default, Internet Explorer has the appropriate settings, so this is a potential issue only for those who have modified their IE security settings.

Where can I get Inquisit 3 Web Edition?

All the components for both Inquisit 3 Desktop and Web are packaged together in a single installer, which can be obtained from our [download page](#). Simply download and install Inquisit 3 and you'll have everything you need to run experiments from the desktop or web. To unlock Inquisit from evaluation mode, you'll need to purchase desktop or web licenses.

How to Run an Inquisit 3 Experiment on the Web

Inquisit 3 Web Edition allows you to launch your experiments directly from a web page. If you have purchased a web license, you have the option of launching experiments from your own web site or from the millisecond.com web site. In either case, data are saved by default to the millisecond.com data service where you can login and download the data files.

If you haven't yet purchased a web license, you can still evaluate Inquisit 3 Web Edition by setting up an experiment on your own web server. When evaluating Inquisit, you can launch and run scripts as normal, but the data will not be saved.

Publishing Inquisit scripts on millisecond.com

Hosting your scripts on millisecond.com is the easiest option for those without experience creating and administering web sites. For those with basic web development skills, this option also includes some support for customizing the launch web page and subject number assignment method. To publish a script on millisecond.com:

1. Write and test your Inquisit script using the Inquisit 3 Desktop Edition editor and tools.
2. Open your web browser and navigate to the millisecond.com web site.
3. Select "My Account" from the menu and click the "Register Inquisit Web Scripts" menu item. If you are not already logged into the site, you will be prompted for your user name and password.
4. Under the "Register Web Scripts" section, click the "Register New Script" link. This will launch the Inquisit Web Script Wizard
5. The first page of the wizard asks whether you wish to host the experiment on millisecond.com or on your own web server. Select the millisecond.com option. Then click the "Browse..." button and select your script file from your local computer. Click next once you have specified the script file.
6. On the next page you can upload additional files used by the script such as pictures and video.
7. Next, select whether you wish to use Inquisit's automatically generated launch page or your own custom web page. The subsequent steps in the wizard allow you specify the title, instructions, and how subject id numbers should be generated and assigned to subjects.
8. When you are done, click the Finish button. That's it, your experiment is now online. You can browse to the launch page using the following url:

[http://research.millisecond.com/\[username\]/\[scriptfilename\].web](http://research.millisecond.com/[username]/[scriptfilename].web)

where [username] is your user id and [scriptfilename] is the original filename of your script.

9. Click the "Start" link to launch your experiment.

Publishing Inquisit scripts on your own web server

Hosting experiments on your own server is an easy if you have access to a web server. To deploy an Inquisit experiment to your web server, follow these steps:

1. Write and test your Inquisit script using the Inquisit 3 Desktop Edition editor and tools, or download a script from the [Inquisit Task Library](#).
2. Navigate to your web scripts page at <http://www.millisecond.com/myaccount/webscripts.aspx>.
3. If the status of your web license is "pending", start your web license by clicking the "Start Now" link.
4. Click the "Register New Script" link to launch the registration wizard and follow the steps in the wizard.
5. On the first page of the wizard, select the option to host the experiment on your own server, and enter the full url to the script file on your server.
6. Continue through the wizard specifying the options you'd like for the launch page.
7. On the final Summary page of the wizard, click the "Download Launch Page" button and save the html page to your computer. Then click the Finish button.
8. Upload your script file and the launch page created above to the location on your web server that you specified when registering the script. If you script uses picture or other media files, be sure to upload those as well.
9. Direct participants to the launch web page to start the experiment.

Gathering Data Over the Web

Inquisit 3 enables you send data from the desktop machine running the experiment to a remote server on the network. The ability to save data to a remote server is critical for experiments conducted over the web because the experimenter often does not have access to the participant's computer to retrieve any locally saved data files. The feature can also be used with Inquisit Desktop Edition, for example, in cases where it is more convenient to save the data to a single location rather than having to copy the data files from multiple computers in a classroom or lab.

There are several options for saving data to a remote server.

1. Save the data to the millisecond.com web server where the experimenter can login and download the files.
2. Send the data to a web server via HTTP POST
3. Send the data to an FTP server
4. Save the data to a shared network (UNC) folder
5. Email the data as a an attachment to an email message.

Of course, it is also possible for both web and desktop experiments to save the data to the local machine in cases where the experiment is run on lab or classroom computers.

Option 1 (saving the data to millisecond.com) is the default behavior of Inquisit Web Edition (IWE), and it is by far the most reliable and easiest solution. To protect against snoopers and sniffers, data is encrypted and posted back to millisecond.com using Secure Sockets Layer (SSL), the same technology used by online shopping and banking web sites to protect sensitive information transmitted over the web. Each participant's data is saved to the server as a separate file. The experimenter can login and download the files from the millisecond.com web site. Again, the downloaded files are encrypted over the network using SSL.

While option 1 is appropriate for the vast majority of experimenters, in some cases it may be necessary or desirable to leverage some of the other remote data features of Inquisit. The means by which data is saved to a server is controlled by the following attributes on the [data element](#).

```
<data>  
/ encrypt = true | false  
/ file = "file path"  
/ password = "password"  
/ userid = "userid"  
</data>
```

Encrypt specifies whether Inquisit should first encrypt the data before being sending it back to the server. If the data is being saved over SSL, this command can be set to "false" since the data will be encrypted using standard web protocols. If SSL is not an option, this command should be set to "true" to obfuscate potentially sensitive data from being compromised by hackers and packet sniffers as it travels over the network. Encrypted data files are saved with the "inq" extension. Unencrypted data files have the "dat" extension.

The *file* attribute specifies the location to which the data is saved. This can be any of the following:

- *Http or https address* (e.g., <https://www.millisecond.com/>). Inquisit uses the HTTP POST protocol to send the data back to the web server. Most web development technologies (PHP, ASP, ASP.NET, JSP) have easy to use methods for extracting data that has been posted to the server.
- *Ftp address* (e.g., <ftp://www.millisecond.com/mydata/>). Inquisit uses the standard FTP protocol to save files to a folder on an ftp server.
- *Unc path* (e.g., \\millisecond\mydata\). If you are running the experiment inside a LAN, you can save the data to a writable network share.
- *Local path* (e.g., c:\inquisit\mydata\). Inquisit saves the data to the specified path on the client computer.

If you specify a folder with no file name, Inquisit will default the file name to that of the script file. Otherwise, it will use the specified file name. So as not to overwrite other data files on the server, Inquisit also appends the date and time (to the nearest millisecond) to the file name along with the "dat" or "inq" file extension depending whether encryption is turned on.

The *userid* and *password* attributes allows you to specify login credentials to use when accessing the web, ftp, unc, or mail server. It is not necessary to specify the userid or password when saving the data to millisecond.com.

When saving data to a remote server, Inquisit creates separate data files for each run of the script as opposed to appending the data to a single file as it does in the case of saving to a local folder. This is to avoid potential collisions that might occur when multiple clients attempt to save to the server at the same time. You can combine multiple data files into a single file by selecting the "Open" command on Inquisit's file menu and multi-selecting all of the files you wish to combine. Inquisit will open all of the selected files, appending them together. You can then select the "Save As" command on Inquisit's File menu to save the combined data into a single file.

Note that data collection capabilities of both Inquisit Web Edition (IWE) and Inquisit Desktop Edition (IDE) are disabled unless a license has been purchased.

Security and Inquisit 3 Web Edition

Running experiments and gathering data over the internet introduces potential security issues that may not exist when running experiments on tightly controlled lab computers. Without taking the proper security precautions, it may be possible for "snoopers" to browse your scripts and data files. This could be a big problem for scripts that contain login credentials, or for data files containing confidential or sensitive data. Malicious hackers could even delete your scripts and data. Obviously, that's not a good thing.

Inquisit has several security features, described below, that enable you to protect your assets.

Protecting your scripts

In order for Inquisit Web Edition to be able to load your scripts from the web, they must be placed on a web server, where it is potentially accessible to anyone with an internet connection. Inquisit supports two strategies for preventing unwanted access to your scripts. First, you can configure the web server to provide access only to authenticated, authorized users. In this case, users will receive a login prompt when Inquisit loads the script, and they must enter a valid userid and password to continue. Second, you can obfuscate the script using Inquisit's built in password encryption system. Anyone who attempts to open the script with Inquisit must first supply the password used to encrypt it. If they attempt to view the script in any other program, they will see nothing but garbage characters.

Authentication Methods

There are a number of different authentication schemes used by web servers to protect content from anonymous access. The following schemes are supported by Inquisit.

Basic Authentication

When Inquisit attempts to load protected content, the server sends a message indicating that authorized credentials are required, and Inquisit will prompt the user for a username and password. If the user supplies credentials, they sent the server, which determines if they are valid and provides access to the content accordingly. Basic authentication works over firewalls, and is widely supported by almost all browsers and web servers. An important caveat is that the userid and password are sent over the network in plain (unencrypted) text, which means they could be read by someone with a packet sniffer. Basic Authentication by itself should not be used for any content that requires real security. However, Basic Authentication used in combination with SSL (otherwise known as HTTPS) is a secure alternative.

Digest Authentication

Like Basic Authentication, when Inquisit attempts to load a script using Digest Authentication, the user must supply a valid userid and password in order to continue. The advantage of Digest Authentication is that the credentials are encrypted before being sent over the network, so they are not vulnerable to packet sniffers. It can also be used to for authentication across a firewall. Digest Authentication is an open standard that is supported by most browsers and web servers.

Integrated Windows Authentication (NTLM)

Windows Integrated Authentication, also known as NTLM, provides another secure way to protect your script. This method protects login credentials by a sophisticated series of handshakes between the client and the web server. Mozilla added support for NTLM in version 1.4, so it is now supported by Firefox and Netscape (as well as IE).

Configuring Authenticated Access to Scripts on the Web Server

To protect your content from unauthorized access, you must place it in a protected folder on your web server. Most web servers have fairly simple setups for protecting a folder. For example, with the Apache web server you can protect the content in a folder using basic or digest authentication by adding a special password file to the folder.

With Microsoft's IIS web server, you can lock down a folder through IIS Manager by right-clicking the folder, selecting Properties, and clicking on the Directory Security tab. From there you can launch the following window. By default, folders allow anonymous access, which means anyone can browse the content. By unchecking this box and checking one of the other options, you can protect the folder.



Encrypting Your Scripts

Encrypting your scripts is another way to prevent unwanted snoopers, whether you are running experiments in a lab or over the web. An encrypted script can not be read unless the user has the password that was used to encrypt it. If you don't have the password, the script is illegible.

Importantly, for web research encryption should be considered a deterrent, not a foolproof security mechanism. The reason is that in order for the Inquisit Web Control to read the encrypted script, it needs to know the password. Thus, you will need to include password in the web page used to launch the file. A clever hacker could figure out that the password specified in the launch page is the same one required to read the script file.

See [Introducing Inquisit 3 Web Edition](#) for details on how to provide the password to the Inquisit Web Control.

Protecting your data files on the server.

By default, Inquisit Web Edition (IWE) saves the data to the millisecond.com web server. IWE uses SSL to encrypt the data before sending it over the wire so that it is protected from network sniffers. Once the data has been saved to the server, you must login to the web server using your Millisecond userid and password in order to access the data files. Without the userid and password, the data files can not be accessed. Using the Millisecond Data Service is the easiest and most secure option for saving data over the web.

Inquisit also supports saving data back to another web server using HTTP POST, or to an ftp server. In either case, you can protect the server (or the particular folder on the server) where you intend to save the data files using basic authentication. You can then specify the userid and password required by Inquisit to access the server in the [data](#) element in your script. If your script contains a userid and password for accessing your data file share, you should always save it as an encrypted inq file rather than a plain text exp file.

Whenever saving data to a web or ftp site, it is good practice to move the files to a more secure location (e.g., not accessible from the internet) as frequently as possible. Ideally, you should write an automated process that moves the files according to a regular interval, or better yet, whenever it detects that a new data file was created.

Alternatively, Inquisit allows you to email the data file to the email address of your choosing. The nice thing about this approach is that you don't have to worry about putting login credentials in your script where someone could potentially see them. Make sure you have enough space in your mailbox and that your spam filter doesn't mistakenly strip out the data file attachments.

If your data files contain sensitive information, you can instruct Inquisit to automatically encrypt using the [encrypt](#) attribute of the [data](#) element. This is a useful privacy mechanism for data gathered over the web or on the desktop. Again, do not forget the password that was used to encrypt the data file, or your data will be lost.

Encrypting Your Scripts

Encrypting your scripts is another way to prevent unwanted snoopers, whether you are running experiments in a lab or over the web. An encrypted script can not be read unless the user has the password that was used to encrypt it. If you don't have the password, the script is illegible.

Importantly, for web research encryption should be considered a deterrent, not a foolproof security mechanism. The reason is that in order for the Inquisit Web Control to read the encrypted script, it needs to know the password. Thus, you will need to include password in the web page used to launch the file. A clever hacker could figure out that the password specified in the launch page is the same one required to read the script file.

See [Introducing Inquisit 3 Web Edition](#) for details on how to provide the password to the Inquisit Web Control.

Protecting your data files on the server.

By default, Inquisit Web Edition (IWE) saves the data to the millisecond.com web server. IWE uses SSL to encrypt the data before sending it over the wire so that it is protected from network sniffers. Once the data has been saved to the server, you must login to the web server using your Millisecond userid and password in order to access the data files. Without the userid and password, the data files can not be accessed. Using the Millisecond Data Service is the easiest and most secure option for saving data over the web.

Inquisit also supports saving data back to another web server using HTTP POST, or to an ftp server. In either case, you can protect the server (or the particular folder on the server) where you intend to save the data files using basic authentication. You can then specify the userid and password required by Inquisit to access the server in the [data](#) element in your script. If your script contains a userid and password for accessing your data file share, you should always save it as an encrypted inq file rather than a plain text exp file.

Whenever saving data to a web or ftp site, it is good practice to move the files to a more secure location (e.g., not accessible from the internet) as frequently as possible. Ideally, you should write an automated process that moves the files according to a regular interval, or better yet, whenever it detects that a new data file was created.

Alternatively, Inquisit allows you to email the data file to the email address of your choosing. The nice thing about this approach is that you don't have to worry about putting login credentials in your script where someone could potentially see them. Make sure you have enough space in your mailbox and that your spam filter doesn't mistakenly strip out the data file attachments.

If your data files contain sensitive information, you can instruct Inquisit to automatically encrypt using the [encrypt](#) attribute of the [data](#) element. This is a useful privacy mechanism for data gathered over the web or on the desktop. Again, do not forget the password that was used to encrypt the data file, or your data will be lost.

Assigning Subject Numbers in Web Experiments

Subject numbers play a number of important roles in experiments. In longitudinal studies, they can be used to correlate data gathered from a subject at different times. For experiments with between-subject variables, subject numbers play a critical role in assigning participants into particular conditions. By using identification numbers such as student ids or telephone numbers, the subject number can also allow researchers to identify participants in cases where the research makes that necessary.

For traditional lab research, managing subject numbers with Inquisit is straightforward. The experimenter starts the Inquisit script and either enters the subject number herself, or she instructs the participant to enter the number. With web research, participants may be located anywhere in the world, in which case there is no experimenter overseeing the data collection session who can assign the appropriate subject number. In these cases, it is necessary to devise a system in which either participants can specify the number themselves, or the number is automatically assigned by the web site. This article will discuss several strategies for assigning subject numbers to web participants.

Subject numbers for between-subject variables

One of the most common uses of subject numbers by Inquisit is to assign subjects into a particular cell of a between-subjects variable. Subject numbers are mapped to conditions within the Inquisit script itself, either by the [expt](#) or the [variables](#) element. For example, in Figure 1 below the script uses the [expt](#) element to counterbalance the order of two blocks across even and odd numbered subjects:

Figure 1.

```
<expt>  
/ subjects = (1 of 2)  
/ blocks = [1=conditiona; 2=conditionb]  
</expt>
```

```
<expt>  
/ subjects = (2 of 2)  
/ blocks = [1=conditionb; 2=conditiona]  
</expt>
```

In most cases, the researcher wants to ensure that subjects are randomly assigned into the even or odd numbered group, and they want equal numbers of subjects in each condition.

Subject number assignment for scripts hosted on millisecond.com

If you choose to host your script on the millisecond.com server, there are several options for generating subject numbers that you can choose from: random generation, random generation without replacement, sequential, user entered, and user entered with confirmation prompt. You simply choose which option you want when running the Script Registration Wizard. There's no need to understand the technical details of how these work.

Subject number assignment for scripts hosted on other servers

If you are hosting the script on your own server, you can select from 3 different sample pages to use as a starting point for your launch page. The sample pages are all linked to from here: <http://www.millisecond.com/web/samplelaunch.aspx>.

Random Assignment of Subject Numbers

The first sample page shows random generation of subject numbers, which is the simplest way to achieve a random, approximately even distribution of subjects into different conditions. Let's take a look at how the reference page randomly assigns subject numbers. (You can see this first hand by browsing to the page and viewing the underlying source.) The first thing to note is the following section of JavaScript at the top of the page source:

In the source code for the page is a javascript method called "GetSubjectNumber" that is responsible for generating the subject number. For random generation, the method looks like this:

Figure 2.

```
function GetSubjectNumber()
{
  return (Math.floor(Math.random() * 1000000000));
}
```

The method contains a single line of code does the work of randomly generating the subject number. The function uses the javascript method , Math.random(), to generate a random number from 0 to 1, then multiplies that value by 1,000,000,000 and rounds it down to the nearest integer so that the final result is a random number between 1 and 1,000,000,000. Note that we are selecting numbers with replacement, so it is theoretically possible that two participants might be assigned the same subject number. However, the chances are slim indeed, and if it does happen, you can use the time of the session as logged in the "date" and "time" data columns to distinguish the subjects' data.

Prompting participants to enter a subject number

For some experiments, it may be necessary or convenient to use personal identification such as a telephone number, social security number, or student id as the subject number rather than an arbitrary, randomly generated number. To administer this kind of experiment over the web, the participant must be allowed to input their id number so that it can be recorded along with the data. The second option allow you to prompt the participant to enter an id number. The third option has the subject enter the number twice to avoid keying errors.

Let's take a look at the javascript for the third option, which is the more complicated method. In this case, the GetSubjectNumber() method contains the following javascript code, which prompts the subject to enter a 5 digit number, then prompts again to confirm the number, and provides error feedback if the number is invalid or they don't match.

Figure 3.

```
function GetSubjectNumber()
{
// This method prompts the subject for an id number and then prompts
  again to // confirm in order avoid mistyped numbers.

var intRegExp = /^(^\d{5,5}$)/;
var promptMsg = "Please enter a 5 digit id number.";
var confirmMsg = "Please confirm the number you entered.";
var invalidMsg = "The number was invalid. Please enter a valid number.";
var matchMsg = "The numbers you entered did not match. Please enter the number again.";

var snum = window.prompt(promptMsg, "");
```

```
while ( snum != null && intRegExp.test(snum) == false )
{
// if the input was invalid, prompt again
  window.alert(invalidMsg);
  snum = window.prompt(promptMsg, "");
}

if ( snum == null )
{
  return null;
}

var sconfirm = window.prompt(confirmMsg, "");
while ( sconfirm != null && (sconfirm != snum || intRegExp.test(sconfirm) == false) )
{
  // if the input was invalid, prompt again
  if ( intRegExp.test(sconfirm) == false)
  {
    window.alert(invalidMsg);
  }

  // if the numbers do not match, alert the user and bail out so
  // they can start over
  else if ( sconfirm != snum )
  {
    window.alert(matchMsg);
    return null;
  }

  sconfirm = window.prompt(confirmMsg, "");
}

return sconfirm;
}
```

The validation is done through the regular expression in the first line of code. You can change the validation by replacing the regular expression (which only verifies that the input is numeric and 5 digits long) with your own. If you don't understand regular expressions, don't worry, the web is abundant with ready made regular expressions for validating all kinds of input (integers, zip codes, telephone numbers, etc.). I found this one in seconds through Google.

Customized subject number generation

If you wish to use your own method for generating subject numbers, you can do so simply by editing the `GetSubjectNumber()` method in the source code. Remember, this javascript code runs on the participant's machine, not on the server, so it has no way to keep track of which subject numbers have already been assigned. For that, you would have to program some server code that tracks subject numbers in a database, then dynamically injects the subject number into this web page, and in particular, into the `GetSubjectNumber()`. If you host your scripts on `millisecond.com`, this is how the hosting service is able to generate sequential subject numbers and random selection without replacement.

These are just a few of the strategies for assigning subject numbers on the web. They are by no means the only techniques possible. Other strategies include deriving subject numbers from the date and time, the ip address of the client computer, or the session id. For those with some familiarity with web development, the web is an flexible and open programming environment that makes any number of schemes possible.

LANGUAGE REFERENCE

see <http://www.millisecond.com/support/docs/v3/index.htm>

E' il momento di ordinare Inquisit 3!

Chiama subito lo 0131.250.313 e parla con un nostro addetto.



Per informazioni commerciali:

NET ENGINEERING
C.so Borsalino 19/A, Alessandria
T.0131. 250313
www.net-eng.it