Creating online experiments in <Frinex>

User Manual for Research Assistants

Max Planck Institute for Psycholinguistics, Nijmegen

Version:
June 2024

Contact:
Thijs Rinsma
thijs.rinsma@mpi.nl
Room 109

Contents

Acknowledgements

This manual would not have been possible without the help of several people. Thanks is due to Peter Withers for his expert advice on Frinex and the Git workflow, to Maarten van den Heuvel for helping establishing the scope of this manual and for proof-reading several chapters, to Dennis Joosen for reading and using earlier versions of this manual and helping to identify several omissions, and to Julia Misersky for making available one of her experiments and stimuli as an example experiment.

PART I: FRINEX BASICS

1 | Introduction

About Frinex

Frinex stands for Framework for Interactive Experiments and is developed at the Max Planck Institute for Psycholinguistics in Nijmegen, The Netherlands. It allows researchers in experimental psychology, language sciences and cognitive sciences to create online experiments. Frinex can be used to create experiments such as sentence rating and picture naming tasks, as well as more complex setups.

Experiments can be made to run in a web browser, on desktop as an Electron app, as an Android or iOS app on mobile and tablets, and as a field experiment (fieldkit).

Timing accuracy

As an online experiment system, Frinex is suitable for experiments that do not have strict timing requirements for stimulus presentation and responses. Although we strive to make Frinex reasonably time accurate for an online system, we cannot guarantee accurate timing. This is because the performance of online experiments depends on things like participants' internet connection, their operating system and the capabilities of their machine (desktop computer, laptop, mobile device) and their screen. In order to judge whether the timing of stimuli is accurate (enough) for your experiment, it should be measured using external equipment (for example, a high-speed camera) on hardware that participants can be expected to use.

Goal of this manual

The present manual is aimed at research assistants. It helps to have a basic understanding of programming and/or scripting, but this is not strictly necessary. The manual covers the use of GitHub Desktop to manage and upload files, explains how to create a both simple and more complex experiments, from the XML definitions to accessing the generated research data.

Keep in mind that experiments can be run using three main versions of Frinex: stable, beta and alpha. Stable is the most well tested (then beta, then alpha). This manual covers the stable version of Frinex as of the time of publication.

Throughout the manual there are several exercises to help you hone your Frinex skills. After having read the manual and completed the exercises, you should be able to use the existing Frinex experiments and the Frinex XML Usage page to discover how to create more complex experiments.

Good luck!

2 | Frinex workflow

In order to create experiments in Frinex you will need a text editor to write XML definitions and a Git client to upload your work. We recommend using an editor that can highlight code, such as Notepad++ or VSCodium[1]. You also need a Git client to upload your work to the Frinex server. As for now, we will explain basic use of Github Desktop, which implements Git in an easy to use graphical interface. You can download it at https://desktop.github.com/

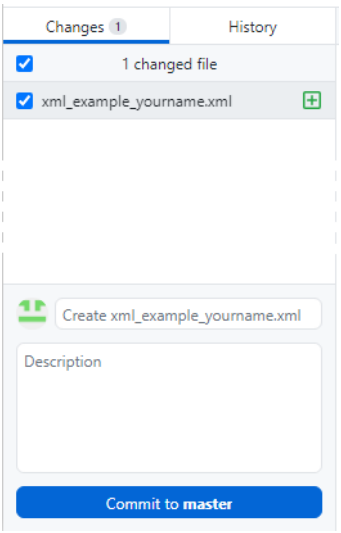
Using GitHub Desktop to upload experiments

The experiments on the Frinex server are organized using Git, which is a widely used version control system (follow the link for more information on version control and Git specifically). It turns a directory on your computer into a so-called 'Git repository' and then automatically tracks changes that are made to files in that directory (the added hidden folder named '.git' is a sign of this). For each Frinex experiment, a repository should contain at least an XML experiment file and a corresponding folder of the same name (without '.xml') for the stimuli.

Creating new experiments and changing existing ones means adding the XML files and changing them; then using Git to communicate this to the repository on the Frinex server. The Frinex server has several repositories for Frinex experiment files, but for now we will be working in our personal repository. Let's go ahead and set up the example experiment that we will be working on in the coming chapter.

Step 1) We first need to create our personal repository:

- Go to http://frinexbuild.mpi.nl/docs/git_setup.html and press 'Calculate Repository URL'. You will be prompted for a user name (full MPI email address) and password.
  Note down the URL you are given.
- In Github Desktop, go to File > Options > Git, fill in your name and email address and click Save.
- Go to File > Clone repository > URL and fill in the URL that was calculated for you.
  At 'Local path', fill in the directory on your computer where you want the repository to be cloned to. You will be prompted again for the user name and password.
- Click 'Clone'

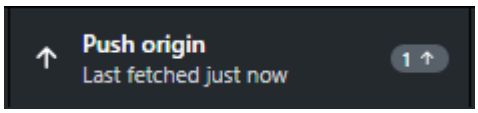


Step 2) Now that we have set up our repository, we can start adding files locally. Let's create a new file called "xml_example_[your name in lower case].xml" inside your local repository. If you go back to Github Desktop, you can see in the left pane that Git has detected the addition of the new file (indicated with a green 'plus' icon).

Although we have added a file, the state of our repository from the perspective of Git has not changed; that is because we still need to *commit* this addition. If you are certain of the change, Write a short description of the change in the provided box and press 'Commit to master'.

[1] We recommend VSCodium in combination with the *XML Language Support by Red Hat* extension. This extension automatically indicates any errors or inconsistencies in the XML.

Step 3) Once the addition of the file is committed, you want to copy the repository in its new state to the server. This is done by *pushing*, using the button 'Push origin' at the top that appears after committing. Be aware that this updates the server repository to be identical to the local repository (i.e. all files in your repository are copied, not just the one you have updated!). Make sure the message reads 'Last fetched just now' before you push:



'Last fetched just now' means that your local repository has been updated with information from the server repository a couple of seconds ago. Therefore, you can safely push, because the only difference between them is the changes you just committed; this is especially important when you work together with others in the same repository (see below note).

After pushing, the new experiment should show up on http://frinexbuild.mpi.nl/ (you need to be connected to MPI network for this to work). This page lists all experiments currently residing on the Frinex server. Order the list by clicking the 'last update' link to bring your newly uploaded experiment to the top. In the 'validation' column, it should say: xml failed This is because you have just uploaded an XML file that Frinex does not know what to do with, as it is an empty file.

After this, working on files and uploading them is a four step process:

- Do a Pull (Repository > Pull)
- Change, add or delete one or more files in your repository
- Commit
- Push

If you want to know more about Git, visit https://www.git-scm.com/

A note on collaboration

When working in the same repository with multiple people, or when working by yourself from different computers, it is especially important to Pull the changes from the remote before you do any work. Say you wait 30 minutes after Pulling and then do a Push. This will replace the information on the server repository with your now-30-minutes out-of-date local repository. Before this Push, someone else may have pushed new changes as well, but this information is now overwritten by your Push. This means the changes they made to their work will be gone from the server repository. When subsequently they decide to Pull (not knowing it no longer contains their changes), they will lose the work they did in their local repository.

That is why it is very important to communicate with your collaborators when you intend to push changes to the remote.

3 | Exploring the XML experiment definition

This chapter is meant to be a quick overview of the structure of an XML experiment definition and use of the Frinex build server. More in-depth information about XML files will be given in chapter 4.

XML basics

Online experiments in Frinex are defined in XML, which stands for “eXtensible Markup Language”. Similar to HTML, XML defines information by means of tags. Unlike HTML tags, XML tags can define any type of information, for example a collection of papers, the inventory of a shop or the structure of an online experiment. See below mock XML file:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<element>
   <nested_element>
   </nested_element>
   <self_closing_element/>
   <element_with_attribute lastElement="yes"/>
</element>
```

The first line (or something very similar) is necessary in any XML file to be recognized as such.

As you can see, elements are defined by an opening statement and a closing statement. The forward slash before the element name makes it a closing statement. Some elements don’t need a separate closing statement and can be closed by putting a forward slash behind the element name.

Elements can be nested inside other elements. The nested element is then called a *child* element and the element it is nested in, a *parent* element. Take note that a nested element must be closed within its parent element. In principle there is no limit to how far the nesting goes – there can be elements nested in elements, nested in elements, etc.

Elements can have *attributes*, which are additional statements behind the element name that inform the way the element functions or that set certain parameters (*lastElement* in the example). The order in which the attributes appear does not matter. But mind the type of quotes you use around the value: these must be straight double quotes ("). Do not use “ , ” , ` or ‘ , as these may not be processed correctly.

Also, note the indentation of the elements nested in <element>. Strictly this is not necessary, but it makes it easier to see the structure of the XML definition.

Finally, you can use <!– and --> to add comments or to (temporarily) deactivate elements. Make sure to avoid double dashes (--) in between a start and end flag. Examples:

```
<!-- single line comment -->
<!--
multi line
comment
-->
<!-- this is -- not -- a correct comment -->
```

Basic structure of a Frinex XML file

Every experiment has stimuli to present, a temporal structure and lay-out for each trial, as well as data being generated by participants as well as internally. Defining what stimuli to present and defining how to present them are done separately in XML. The below schematic shows the basic elements that make up an experiment definition. The <deployment>, <scss>, <metadata> and <stimuli> elements have fixed places with respect to each other.

```
<experiment>          start of the XML definition

    <deployment>      tells Frinex how to deploy the experiment.

    <scss>            allows you to define CSS formatting styles in your experiment (optional)
    </scss>

    <metadata>        allows you to define metadata fields

    <presenter>       the central building blocks of your experiment. Use as many of them as you
                      need to create the screen contents and temporal structure (see Figure 2)

    <stimuli>         here you define the stimuli to be presented in your experiment.

</experiment>         end of the XML definition
```

Figure 1: Order of child elements in main <experiment> element and what they are for

The elements we are concerned with at the moment are <metadata>, <presenter> and <stimuli>.

The <metadata> element is where you define data fields that can contain various sorts of information. Each field is created by using the <field> element. Most experiments need fields like this, for example for demographics information that you want to collect or for control switches in complex setups.

Every presenter needs two attributes: *self* and *type*. The name (*self*) of the presenter uniquely identifies the presenter, which allows you to refer to it from other presenters. The presenter's *type* determines for what purposes the presenter is best suited. A "menu" presenter is used for things like the welcome screen and informed consent screen, "stimulus" presenters allow you to define the trial structures in your experiment and "transmission" presenters are used to send the data collected during the experiment to the Frinex database.

Finally, the <stimuli> element contains stimulus definitions in the form of <stimulus> elements, usually one for each trial. A stimulus presenter uses the stimulus definitions from <stimuli> to know what stimuli to present from one trial to the next.

Below is a summary in schematic form:

```
<experiment ...>

   <deployment .../>

   <scss>
   </scss>

   <metadata>
      <field .../>
      <field .../>
   </metadata>

   <presenter self="welcome" type="menu">
      <plainText featureText="Welcome!"/>
   </presenter>

   <presenter self="demographics" type="metadata">
      <metadataField .../>
      <metadataField .../>
   </presenter>

   <presenter self="mainTrials" type="stimulus">
      <loadStimulus>
        ...
      </loadStimulus>
   </presenter>

   <presenter self="admin" type="transmission">
      <sendMetadata>
        ...
      </sendMetadata>
   </presenter>

   <stimuli>
      <stimulus .../>
      <stimulus .../>
      <stimulus .../>
      ...
   </stimuli>

</experiment>
```

Define metadata fields

Present input boxes

Present stimuli

Send data

Define stimuli

An example experiment in action

The Frinex server contains experiments including their source files. As mentioned before, these experiments are listed on the Frinex build page at http://frinexbuild.mpi.nl. For now, we are concerned with four of the columns on the page: the experiment name ("experiment"), the date and time of last update ("last update"), links to XML files ("validation") and the links to the staging version of the experiment ("staging web").

Navigate to the experiment name "xml_example". The list can be ordered by name by clicking the experiment link at the top of the column. The "validation" column for xml_example says xml failed, which means something is wrong with the XML file and the experiment has not been built. Click the failed link to see the error message associated with this. To see the contents of the XML file, click on the xml link and on the next page, right-click in the page and select 'View page source'. Copy all of the text that appears and paste it into your preferred text editor (see Chapter 2).

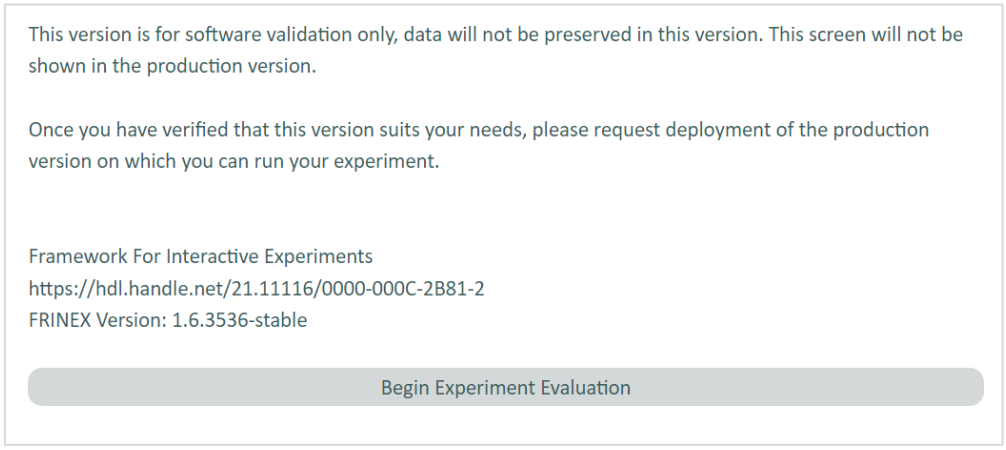
Exercise 1: fix the XML file

Use your knowledge of XML and Frinex to build your personal "xml_example" on the Frinex server. To do this, paste the XML code into an editor and save it as "xml_example_[your name].xml". Make the necessary fixes. Next, upload this file to the Frinex server using Git.

TIP: **on the build page you can order the experiments by upload date by clicking the 'last update' header.**

TIP 2: **use the XML extension for VSCodium / VSCode to make your life easier.**

Once you have a working XML file, the 'validation' column will show xml passed and the 'staging web' column will show building. When it is ready, the 'staging web' column will show four links. Click the browse link to go to the experiment. You will first see the screen below.



Click on "Begin Experiment Evaluation" to start the experiment.

As you have seen, this example experiment displays only text and does not require any input from the participant during the trials. We will get to that in the next chapter. First, let's relate what you see in the experiment to the XML structure.

The first screen you see is the welcome screen, which corresponds to the first presenter in the XML file, called "welcomeScreen". The welcome message itself is produced by <plainText>. There is also a "Next" button spanning the width of the screen, created by the <targetButton> element. When clicked, Frinex goes to the presenter indicated in the *target* attribute, which needs to correspond to the *self* attribute of the target presenter ("demographics" in this case).

The second screen shows you two boxes for text input which correspond to the <metadataField> elements, as well as a button corresponding to <saveMetadataButton>. The input boxes are coupled to the <field> elements in <metadata>, each of which needs the following attributes:

postName:	The name of the data field
registrationField:	The text displayed above the input box
controlledRegex:	The allowed values for this data field in the form of a regular expression ('regex')
controlledMessage:	The message displayed if the value deviates from the allowed values

The button will check whether the fields and their values in <metadata> could be successfully saved and they are also sent, because the *sendData* attribute is set to "true". If so, Frinex will trigger <gotoNextPresenter> which takes the experiment to the presenter specified in the attribute *next* of the current presenter.

Clicking the 'Start experiment' button takes us to the part where the trials are presented to the participant. This is done using a presenter of type 'stimulus', which has the following basic structure:

```
<presenter type="stimulus">
   <loadStimulus>

      <hasMoreStimulus>      Triggers elements for each stimulus in succession.
      </hasMoreStimulus>

      <endOfStimulus>        Triggers elements once after the last trial has finished.
      </endOfStimulus>

      <randomGrouping>       Provides additional control over stimulus selection.
      </randomGrouping>

      <stimuli>              Contains tags that determine which stimuli are selected for
      </stimuli>             presentation. Not to be confused with the main <stimuli>
                             element within <experiment>

   </loadStimulus>
```

The presentation of each stimulus depends on what you put in <hasMoreStimulus>. First, the screen is cleared with <clearPage/> to clear away the previous stimulus. In the above example, some text is presented using <stimulusLabel>. The text to be displayed is taken from the *label* attribute of a <stimulus> element.

```
<hasMoreStimulus>
    <clearPage/>
    <stimulusLabel styleName="labelStyle"/>
    <pause msToNext="1500">
       <nextStimulus repeatIncorrect="false"/>
    </pause>
 </hasMoreStimulus>
```

We want the label to be bigger and to appear centered on the screen, so we apply the CSS style called "labelStyle", which is defined in <scss>. As Frinex does not automatically continue to the next trial, we tell it explicitly with <nextStimulus>. But it must first wait for a certain duration, otherwise the stimuli will be presented too rapidly. To do that, we embed <nextStimulus> within a <pause> element, which defines the length of the pause in the *msToNext* attribute[2].

In our XML file, <endOfStimulus> contains the element <gotoNextPresenter>. This means that, after all the stimuli are presented, Frinex will go to the presenter with name 'admin' (as defined in the *next* attribute of the stimulus presenter). Alternatively, you could use <gotoPresenter> and use its *target* attribute to set the target presenter manually.

The last presenter in the XML file does two things: it displays the message "this is the end of the experiment" and it sends all the data to the Frinex server. The <onSuccess> and <onError> elements in <sendMetadata> are triggered either when Frinex successfully sent the data, or when it failed to do so, respectively.

Exercise 2: change the experiment

- The stimuli appear to be presented in the wrong order; see if you can fix this.
- Increase the speed of stimulus presentation; it is going rather slow.
- Change the experiment so that participants are asked to fill in information after the trials are done instead of before.
- Add a thank you message at the end of the experiment and make it stand out.

[2] The value of *msToNext* is in milliseconds, but this does not mean it is accurate on the millisecond level.

Clearing your browser's cache

When working on an XML file and going back and forth between it and your experiment, your browser may hold on to the data of a previous version of the experiment. This can prevent you from seeing the changes you applied to your XML in the experiment. To make sure your browser uses the newest data, you can do two things:

- Add a debug presenter just before the main <stimuli> element in the XML file:

```
<presenter self="about" menuLabel="debug" type="debug" title="debug">
    <stimuliValidation />
    <versionData />
    <eraseLocalStorageButton />
</presenter>
```

  This presenter allows you to clear application data; just add "/?debug" to the URL of the experiment. So for example, frinexstaging.mpi.nl/my_experiment/?debug. This will take you to a screen with a button 'Erase Stored Data'. Clicking this will erase the stored data of this experiment and return you to frinexstaging.mpi.nl/my_experiment.

- Sometimes doing this is not enough; in that case you can also force refresh the webpage (also called 'bypassing the cache'). See https://en.wikipedia.org/wiki/Wikipedia:Bypass_your_cache for how to do this on different OS's and browsers.

4 | Defining a simple experiment in XML

It is time for a more thorough look at the definition of a complete experiment. This includes some basic settings in the XML file, user input, trial types, picture and sound presentation, as well as some useful techniques for formatting screen contents.

We will be using the experiment called “picture_naming_example”, which you can find on the Frinex build server. Click the “xml” link, copy the contents to a new XML file of your own and rename it “picture_naming_example_[your name in lower case]”. Then, download the files used in the experiment from the online experiments page on Maxintra

From time to time we will refer to the Frinex XML Usage page. This page lists all XML elements and their attributes and provides a description for many (but not all) of them. Use your browser’s search function to look up any element or attribute you want to know more about.

The <experiment> element

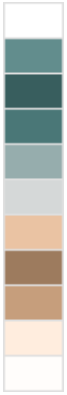
Before we dive into the detail, let’s consider few important attributes in the main <experiment>:

- *xmlns:xsi* and *xsi:noNamespaceSchemaLocation*: these are needed to tell Frinex how to interpret the XML and its elements (although they do not show up on the XML Usage page). You don’t need to change anything to the first one. The second one sets the version of Frinex to use and should normally be set to “stable.xsd”. As the name implies, stable is the version that has been tested for longest and is considered to be the most robust. Beta comes next, which you might use if you need functionality that is not yet available in the stable version. The alpha version is not recommended, as it contains features that are not well tested.

- *appNameDisplay*: this refers to the name displayed in the browser.

- If you want to adjust the text and background colors used in the experiment, the below attributes allow you to do this. The values represent the RGB color components and can be inserted into any drawing / sketching app to test the color. The background color determines which of the primary and complement colors are used. Below values are the default values:

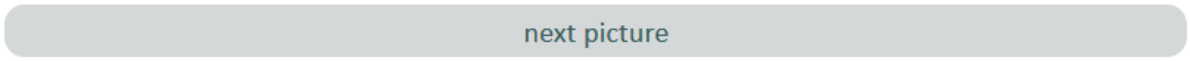
backgroundColour="#FFFFFF"
primaryColour0="#628D8D"
primaryColour1="#385E5E"
primaryColour2="#4A7777"
primaryColour3="#96ADAD"
primaryColour4="#D5D8D8"
complementColour0="#EAC3A3"
complementColour1="#9D7B5E"
complementColour2="#C69E7C"
complementColour3="#FFEDDE"
complementColour4="#FFFDFB"



User input with buttons

In Frinex you can create several types of buttons for participants to use. Below we shortly show you how to implement them in your XML file, how they appear in the browser and explain what to keep in mind when using them.

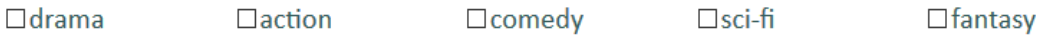
```
<actionButton featureText=”next picture”>
```




```
<ratingButton ratingLabels="none,a few,several,many" ratingLabelLeft="left" ratingLabelRight="right"/>
```


```
<ratingRadioButton ratingLabels="1,2,3,4,5"/>
```


```
<ratingCheckBox ratingLabels="drama,action,comedy,sci-fi,fantasy"/>
```



The **actionButton** is the basic button. Whatever elements are nested within this element, will be triggered once the button is pressed. For example:

```
<actionButton featureText="next picture">
    <pause msToNext="1000">
       <nextStimulus repeatIncorrect="false"/>
    </pause>
 </actionButton>
```

In the above example, when the button is pressed, Frinex waits 1 second and then goes to the next stimulus. Suppose you want no delay, then for convenience you could use <nextStimulusButton/>, which does the same as <actionButton> and <nextStimulus> combined. Also, action buttons can detect keystrokes by adding the attribute *hotkey* (see the XML usage page for allowed *hotKey* values). An alternative for this button is stimulusButton, which works in the same way as actionButton, but in addition it automatically logs button presses on the Frinex server.

The **rating buttons** and their different forms give you one or more buttons in a row. The <ratingButton> and <radioButton> allow for only one choice among the options, while the <ratingCheckBox> allows for multiple choices. The *ratingLabels* attribute is mandatory.

Another useful button is the **redirectToUrl** button. This places a button that takes the participant to the website specified in the ‘src’ attribute. This is useful for taking participants to an additional survey or to a site such as Prolific to confirm their participation when the experiment has ended.

User input with text fields

Besides buttons, there are text fields for participants to enter text. We have mentioned <metadataField>, which saves the entered text in a data field predefined in <metadata>. or you can use <stimulusFreeText>, which saves the text as data tied to the current stimulus:

```
<stimulusFreeText featureText="Use at least 3 characters" validationRegex=".{3,}" inputErrorMessage=""/>
```

The displayed text box will accept only a single line of text by default; for multi-line input, make sure the value for *validationRegex* is "[\\s\\S]*" (zero or more characters) or "[\\s\\S]{1,}" (one or more characters).

Whatever is written into the text box will count as input as soon as participants go to the next trial. Note that the attributes *featureText*, *validationRegex* and *inputErrorMessage* are mandatory. Contrary to what you might expect, here *featureText* is the message that is displayed when the input does not conform to the regex specified in *validationRegex*. The value in *inputErrorMessage* will be shown if characters are used that are not in *allowedCharCodes* (*inputErrorMessage* should be present, but it is okay if no value is assigned to it).

For more details on stimulusFreeText, see the XML Reference.

Presenting pictures and sounds

In order to present picture and sound stimuli, you need to upload the files to the experiment folder on the server. Simply add them to the experiment folder in your local repository and push this change, just as you would push any change to your XML file. For audio files you can use three formats: OGG, MP3 and WAV. Only use WAV as a last resort, as this is not optimized for the web.

Images and sounds can be presented by using the elements <stimulusImage> and <stimulusAudio> within a presenter of type stimulus.

<stimulusImage> has two mandatory child elements: <mediaLoaded> and <mediaLoadFailed>. If the image is succesfully loaded, any elements within <mediaLoaded> will be triggered. If not, any elements within <mediaLoadFailed> will be triggered. There is also a mandatory attribute *msToNext*, which is the time the image is visible for.

```
<stimulusImage msToNext="1000">
    <mediaLoaded>
    </mediaLoaded>
    <mediaLoadFailed>
    </mediaLoadFailed>
 </stimulusImage>
```

If you want to present an image that is not defined in the main <stimuli> element, you can use <image> instead of <stimulusImage>. The only difference is that you need to specify the path name of the image file using the *src* attribute:

```
<image src="fixation_cross.png" msToNext="1000">
    <mediaLoaded>
    </mediaLoaded>
    <mediaLoadFailed>
    </mediaLoadFailed>
 </image>
```

As for presenting audio, there is the <stimulusAudio> element, which has four mandatory elements: <mediaLoaded>, <mediaLoadFailed>, <mediaPlaybackStarted> and <mediaPlaybackComplete>. The first two function in the same way as in <stimulusImage>; the last two become active when the sound has started playing or has stopped playing, respectively. For example:

```
<actionButton featureText="Play sound">
    <stimulusAudio autoPlay="true" mediaId="soundID" showPlaybackIndicator="false">
     <mediaLoaded/>
     <mediaLoadFailed>
        <htmlText featureText="Failed to load audio"/>
     </mediaLoadFailed>
     <mediaPlaybackStarted/>
     <mediaPlaybackComplete>
        <nextStimulus repeatIncorrect="false"/>
     </mediaPlaybackComplete>
    </stimulusAudio>
 </actionButton>
```

This will play a sound once the button is pressed, and go to the next trial when the sound has finished playing. The attributes *autoPlay*, *mediaId* and *showPlaybackIndicator* are mandatory.

For Frinex to use the picture and sound files you uploaded, you must refer to them in every <stimulus> element at the end of your XML file. For pictures this means adding the attribute *imagePath* and for audio this is the attribute *audioPath:*

```
<stimulus identifier="trial1" imagePath="plant.png" audioPath="sound”/>
```

Note that the value for the *audioPath* attribute should not include the file extension (unlike the value for the *imagePath* attribute).

Exercise 3: trial structure

- Each image currently stays on the screen during the next trial, forcing the next picture to appear below it. Make sure to first clear the picture from the previous trial from the screen before presenting a new one.

- One image cannot be loaded; make sure it can be.

- The current experiment shows a couple of pictures, but the repository contains twenty of them. Make sure all pictures are presented.

- Based on the following requirements, create the trial structure for this experiment.
    - Present a fixation cross for 1000 ms (don't worry about it not being centered in the middle of the screen)
    - Present picture for 500 ms
    - Then show an input box with the text "The picture showed..." above it. Make sure the input box requires at least 3 characters and asks for them if not given. Also add a button for going to the next trial below the input box,

Basic formatting

In the picture naming experiment, participants now see a picture, can respond with text input and go on to the next picture until all trials are done. But the screen contents are crammed at the top, because we haven't told Frinex how to position the contents.

Basically, there are two ways to do this. The first method is to use built-in formatting elements:

- `<centrePage/>`
  This element horizontally centres all screen contents.
- `<addPadding/>`
  This adds an empty line, useful for separating items vertically. As long as you have a simple screen layout for your trials, you can separate the text and image elements using only <addPadding> elements; just add as many as you need.

The second way to lay out the screen contents is by using CSS styles. This way you can control things like position, color and size of any element that has an attribute *styleName.* You can set the value of this attribute to the name built-in styles (see below) or to a custom style you defined yourself. The latter allows for more control but requires knowledge of CSS (see chapter 7).

For now, here are some styles that are built into Frinex that you can use right away:

- **centeredVerticalHorizontal** centers the element in the middle of the screen, both vertically and horizontally. The element will stay in the centre even when scrolling.
- **footerCenteredHorizontal** centers the element only horizontally and puts it at the bottom of the screen.
- **minimalWidth** horizontally condenses the item to no more than 500 pixels.
- **fullScreenWidth** fits the item horizontally to 100% of the window's width.

Regions

If you use a style on an element, the style is applied only to that element. For example, an image, an input box and some text with the style 'centeredVerticalHorizontal' will all have that style applied separately. The consequence is that they will all be put in the centre of the screen on top of each other. This can be avoided by putting the elements in a region and applying a style to the region, which will centre the elements as a group:

```
<regionAppend regionId="stimulusRegion" styleName="centeredVerticalHorizontal">
   <stimulusImage>
   <stimulusFreeText>
   <htmlText>
</regionAppend>
```

Stimulus randomization

By default Frinex will present the stimuli defined in <stimuli> from the top down. To randomize stimulus presentation, set the *randomise* attribute of <loadStimulus> to "true". After randomization, there may be adjacent stimuli that are the same in some way. You can then use the *adjacencyThreshold* attribute to set the window of disallowed proximity; a value higher than 0 will modify the order in case a stimulus is surrounded by matching stimuli (default is 3). Whether stimuli match is determined by the values of the <stimulus> attributes *image, video, audio* or *label*, whichever comes first in the <stimulus> element.

See the XML Usage Page for more details on randomization.

Stimulus selection

All stimuli will be used for presentation, as long as you do not discriminate between them. Many experiments of course do make a distinction, for example between stimuli in different experimental conditions. To do this in XML, stimuli belonging in a group need to be tagged as such both in the presenter and in the <stimulus> elements. In the presenter, put the tag name in the <stimuli> element of <loadStimulus>. In the <stimulus> element, put the tag name in the *tags* attribute:

```
<presenter>
    <loadStimulus>
     <hasMoreStimulus/>
     <endOfStimulus/>
     <randomGrouping/>
     <stimuli>
        <tag>practice</tag>
     </stimuli>
    </loadStimulus>
 </presenter>
 ...
 <stimuli>
    <stimulus identifier="trial" imagePath="plant.png" tags="practice"/>
 </stimuli>
```

You can also randomly assign participants to different stimulus lists / groups. To do this, let Frinex randomly select a stimulus tag, for example “group1” or “group2”, by using the <randomGrouping> element in <loadStimulus>.

For example:

```
<presenter type="stimulus">
    <loadStimulus>
       <hasMoreStimulus>
          ...
          <randomGrouping>
             <tag>group1</tag>
             <tag>group2</tag>
             <tag>group3</tag>
          </randomGrouping>
       <hasMoreStimulus>
    </loadStimulus>
 </presenter>

 <stimuli>
    <stimulus identifier="g1_trial1" tag="group1"/>
    <stimulus identifier="g1_trial2" tag="group1"/>
    <stimulus identifier="g2_trial1" tag="group2"/>
    <stimulus identifier="g2_trial2" tag="group2"/>
    ...
 </stimuli>
```

Exercise 4: formatting and stimulus control

- Center the image, the fixation cross, the message “The picture showed”, the input box and button in the middle of the screen. Make sure the message is displayed above the input box.
- Include practice trials in the experiment; make sure only practice trials are selected for the ‘practice’ block and only main trials are selected for the ‘main’ block. Include a message after the practice trials informing that the experiment is about to begin.
- Randomize the presentation of the main trials, but not the practice trials.
- Make Frinex randomly select either pictures of things you can eat or pictures of things you cannot eat.

5 | Let the experiment begin!

Publishing an experiment

The example experiments you have made so far were built on the **staging server**. This is the part of the server where experiments are developed and tested before they are finally published.

If you have thoroughly tested an experiment and want to run it with participants, you should publish it on the **production server**. To do this, change the *state* attribute of the <deployment> element from "staging" to "production". When you re-upload the experiment, it will go through the normal build process and will continue to build the production version after it has finished building the staging version.

```
<deployment     publishDate="2021-15-05"
                expiryDate="2021-15-11"
                isWebApp="true"
                isDesktop="false"
                isAndroid="false"
                isiOS="false"
                state="production"/>
```

When you no longer need to use your experiment, you can undeploy it. To do this, change the *state* attribute to "undeploy", then re-upload the experiment. Also, if you wish to make a last minute change to an experiment that is already published, first undeploy it, then make the necessary changes and finally redeploy it by changing the *state* back to 'production'.

Accessing the data

Every experiment has a corresponding admin page from where you can access the research data:

http://frinexstaging.mpi.nl/[name of experiment]-admin (for the staging server)
http://frinexproduction.mpi.nl/[name of experiment]-admin (for the production server)

To access the data for staging experiments, you can log in using the name of the experiment as the user name and ask for the password by emailing Peter Withers or Thijs Rinsma. This password is the same for all staging experiments. To access the data for a production experiment, you need a unique password for the experiment in question. You can get it by clicking the 'access' link in the 'production admin' column on the Build page. Provide your MPI login credentials and the user name and password will appear on your screen.

Once logged in, you can access the data from multiple sources:

1. View the data directly from the browser using the links under 'View data' (click the link in the upper right to show all information)
2. Download separate data files using 'Download Zipped Data' and 'Download Zipped Audio'

Both sources will provide you with the following information:

View data / Zipped data file	Information
Group Data Viewer / groupdata.csv	Info about participant groups if applicable
Participant Listing / participants.csv	Demographics and other metadata
Screen Events Viewer / screenviews.csv	What screens were displayed in the experiment and
Stimulus Response Viewer / stimulusresponse.csv	Logs on stimuli presentation and participant input
Experiment Data Viewer / tagdata.csv	Frinex version, browser and operating system used by every participant
Experiment Data Pair Viewer / tagpairdata.csv	Similar to stimulus responses .csv, but in a different form
Time Stamp Viewer / timestampdata.csv	Data on timing of image presentation and participant input, as well as occurrences of <logTimeStamp>
Audio Listing / audio download links	Recorded audio files

All CSV files contain a unique user ID for each participant and the date and time on which the data was logged (but date and time are not in participants.csv). Sometimes the data viewer shows a few extra columns where they don't exist in the corresponding CSV file (for example 'ScreenName').

The trial data are logged in stimulusresponse, tagpairdata and timestampdata. All of them contain timing information of events, although in general timestampdata contains the most accurate time values. The numbers are relative to the start of the presenter which the logged event is part of. Be aware that all timing data reflect the internal registration of events. Although they give an indication of the timing of events displayed on screen or through speaker, this should be measured with video recording / optic sensor or microphone to know for sure.

Responses made with a <stimulusButton> and text presented with <stimulusLabel> are automatically logged in timestampdata.csv as 'stimulusButton' and 'stimulusLabel_out'.

You can also log custom timestamps at any moment during the experiment:

```
<logTimeStamp eventTag="picture_presented"/>
```
eventTag will be logged in timestampdata.csv

```
<logTokenText type="stimulusTiming" headerKey="Picture" dataLogFormat="picture_presented"/>
```
type will be logged in column 'EventTag'
headerKey will be logged in column 'TagValue1'
dataLogFormat will be logged in column 'TagValue2'

PART II: ADVANCED TOPICS

In the first part of this manual we gave an overview of how to create simple experiments, from writing the XML file and uploading it using Git, to putting the experiment in production and accessing the experiment data. The second part of this manual aims to help you create experiments with more complex features. It does not (yet) contain exercises.

6 | Recording audio

During an experiment you can record audio using the browser in which the participant is doing the experiment. The following setup inside the <hasMoreStimulus> element is a minimal audio recording example:

```
<startAudioRecorderWeb mediaId="audio_recording"
                       featureText="Now recording"
                       downloadPermittedWindowMs="0">
   <onSuccess>
      <pause msToNext="5000">
         <stopAudioRecorder/>
         <nextStimulus repeatIncorrect="false"/>
      </pause>
   </onSuccess>
   <onError>
      <htmlText featureText="Could not start the recorder"/>
   </onError>
   <mediaLoaded/>
   <mediaLoadFailed>
      <htmlText featureText="Failed to upload recording"/>
   </mediaLoadFailed>
   <mediaPlaybackStarted/>
   <mediaPlaybackComplete/>
</startAudioRecorderWeb>
```

This will try to start the browser's audio recorder, and if successful, records for 5 seconds. After that, it stops the audio recorder with <stopAudioRecorder> and goes to the next stimulus. If anything goes wrong with the audio recording, <onError> is triggered and an error message appears. It is probably wise to add a button that allows the user to try again or do something else.

It could also be that the recording succeeds, but that the file cannot be uploaded to the server. In that case, <mediaLoadFailed> will be triggered. If it is uploaded successfully, <mediaLoaded> will be triggered.

During recording a red indicator in the top right corner of the screen is displayed. It will show the message set in *featureText.* You can turn it into a running timer with *featureText*="00:00:00".

The default file format used to save the recording is **.ogg**. You can change this to **.wav** format if you need to by adding the attribute *recordingFormat*="wav".

If you want to give feedback about the sound level during the recording, you can set *levelIndicatorStyle* to a CSS style name; an empty "" will result in a very basic indicator. In order to position it, append a region in the presenter with *regionId*="AudioRecorderWebLevelIndicator" and a value for *styleName.*

Playing back recorded audio

To play back the sound that was recorded, first set the allowable time window for playback using *downloadPermittedWindowMs.* That is, after the audio is recorded, it is uploaded to the Frinex server. In order to play it back, it must then be downloaded. The time window allowed for this should be limited for security reasons. The window starts when the recording is uploaded. For really long audio recordings (minutes) it is wise to make this window longer than for short ones (seconds), especially if the file format is .wav (larger than .ogg so takes longer to download).

In order to trigger the playback of the recording, use the <playMedia> element. It will play the audio that has a matching *mediaId*, as long as the time window set in *downloadPermittedWindowMs* has not yet passed. If the playback fails then mediaLoadFailed will be triggered.

As an example, you could add <playMedia> in <onSuccess> to play back the recording right away using a button:

```
<onSuccess>
    <pause msToNext="5000">
       <stopAudioRecorder/>
       <actionButton featureText="play">
          <playMedia mediaId="audio_recording"/>
       </actionButton>
    </pause>
 </onSuccess>
```

7 | Formatting and layout

In Chapter 4 we covered the basics of formatting and layout of screen contents. To recap, there are two ways: using predefined elements and using CSS styling. In this chapter we will cover these methods and the use of regions in more detail.

More layout elements

- There are variants of button elements that will put the button at the bottom of the page: `<actionFooterButton>`, `<targetFooterButton>` and `<ratingFooterButton>`
- `<table>`
  A table lets you organize screen items into rows and columns:

```
<table styleName="">
    <row>
       <column styleName="">
          <htmlText featureText="left column"/>
       </column>
       <column styleName="">
          <htmlText featureText="right column"/>
       </column>
    </row>
 </table>
```

  Note that the first element in a table should always be a row; from there you can add more columns, which can contain rows, etc. You can apply styles to the table as a whole and to individual columns as well. These styles may or may not combine very well when used simultaneously. Also, be aware that using different amounts of contents (some or a lot of text, a small or a large picture) will cause the rows or columns that hold them to change in size accordingly.

CSS styling

To use custom CSS styles you need to define them within the <scss> element as shown below. There are plenty of resources online detailing which properties and values are available, see for example https://developer.mozilla.org/en-US/docs/Web/CSS/Reference

```
.nameOfStyle {
   property1: value;
   property2: value;
   ...
}
```

In-line CSS

Besides applying a style to an element, you can add styling directly in the *featureText* of <htmlText> and <htmlTokenText>. Basically, this means adding HTML code to the *featureText*. This makes it easy to change the color of some words in a text, for example, but adding an image is also possible. Here are some examples:

```
<htmlText featureText="<center>This text is centered.</center>"/>
```

```
<htmlText featureText="<span style="color: blue;">This text is blue.</span>"/>
```

```
<htmlText featureText="<div><img src="logo-max.png" width="400" align="left"/></div><br/>"/>
```

We cannot use < and > inside *featureText,* because they are already used for the XML elements. Instead we need to replace them with `<` and `>` (less than / greater than). We also need to use `&quot;` instead of double quotes (") for the same reason. So the first example becomes:

```
<htmlText featureText="&lt;center&gt;This text is centered.&lt;/center&gt;"/>
```

8 | Advanced Control

Tokens

In an XML definition you can reference the value of a metadata field and use it as part of attribute values in other elements. Such a reference is called a 'token'. Tokens either take the form of ::fieldname:: when referring to built-in metadata fields, or ::metadataField_fieldname:: when referring to metadata fields you have defined in <metadata>. Appendix B details the available tokens, but here is a simple use case:

Say you want to display how far into the experiment participants have progressed, based for example on the trial number. For this you could use the element <progressIndicator>:

```
<progressIndicator evaluateTokens="(::stimulusCode:: / N) * 100">
    <onSuccess/>
    <onError/>
</progressIndicator>
```

The attribute *evaluateTokens* needs a dynamic value that changes as a function of the trial number. For this we use the token ::stimulusCode::, which refers to the value of the *code* attribute in <stimulus> elements. The progress indicator interprets the value of *evaluateTokens* as a percentage. If we number our trials using the *code* attribute, we can make the progress bar indicate the percentage of trials finished, based on the total amount of trials N.

You can try out the progress indicator element in with_stimulus_example. Press any of the percentage buttons to show the progress in the bars. The default is the upper most bar, but you can see how to create the other styles in the XML of with_stimulus_example (search for "progressIndicator" and then look up the used styleName in the <scss> section of the XML).

Randomization by Frinex

Besides enabling stimuli randomization by setting the *randomize* attribute of <loadStimulus> to "true", we can use the following attributes for more control over the randomization:

repeatAmount
How many times the (randomized) stimuli will be repeated after having been presented once.

repeatRandomWindow
This randomizes your stimuli in a moving window of the size that you indicate, after first applying the repeat count. The moving window goes through your stimuli in steps of 1, and ranges from the current stimulus up to the size you indicated. A number equal to the total number of stimuli will make it randomize all items to any location.

It is possible to try out different combinations of these values in your experiment using the transition table calculator. This provides information about the probability of adjacent stimuli and the like. Add the following code to your experiment:

```
<presenter self="about" type="debug">
    <versionData/>
    <stimuliValidation/>
 </presenter>
```

Then you can access it by appending '#about' to the experiment URL, for example https://frinexstaging.mpi.nl/my_experiment/#about

Using multiple stimulus lists

If you want to present different groups of participants with predefined lists of stimuli, there are two ways. Both of them require a metadata field in <metadata> to store the list names. For example:

```
<field controlledMessage="." controlledRegex=".*" postName="stimulus_list"
registrationField="stimulusListID"/>
```

The first method builds on the random group selection method from Chapter 4, section Stimulus Selection. Once you have that set up, add the attribute *storageField* to <randomGrouping> and give it the same value as the *postName* of the metadata field above.

This allows you to select which list your participants will be presented with, either by setting it with <setMetadataValue *fieldName*="stimulus_list"> or by adding a GET parameter in the link to the experiment, for example:
https://frinexstaging.mpi.nl/my_experiment/**?stimulusList=list2**

The downside of this method is that your XML becomes very large if you have many trials and many lists. It may even be impossible to build your experiment, as the maximum number of lines in XML is somewhere in the range 5000 to 7000.

To avoid this downside, you can do some things differently:

- Use the <list> element in <randomGrouping> to define the order of stimuli in each list. You do this by stringing together the identifiers of every <stimulus> in the desired order, like so:

  ```
  <randomGrouping storageField="stimulusList">
     <list alias="list1">-trial1-trial2-trial3- ... -</list>
     <list alias="list2">-trial3-trial5-trial2- ... -</list>
     <list alias="list3">-trial7-trial2-trial9- ... -</list>
  </randomGrouping>
  ```

- There is no need to add the list names to the <stimulus> *tags* attribute.

Triggering a block of elements

You can trigger a block of multiple elements, by first defining the block of elements inside <triggerDefinition> and then using <triggerMatching> to trigger them:

```
<triggerDefinition listenerId="event" threshold="1" maximum="1">
    (block of elements)
</triggerDefinition>
...
<triggerMatching listenerId="event"/>
```

By using <triggerMatching>, you make a trigger request and it takes the number of requests set in <triggerDefinition> *threshold* to actually trigger the block of elements. By setting *maximum* higher than 1, you allow the block to be triggered more than once. Make sure the *listenerId* attributes of <triggerDefinition> and <triggerMatching> correspond.

Setting metadata field values

You can set the value of a metadata field using below methods:

- `<setMetadataValue fieldName="" dataLogFormat="">`

  Here, *dataLogFormat* is the desired value of the metadata field named in *fieldName.* Optionally, you can use the attribute *replacementRegex* to change the value of the metadata field using a regex. This is only useful if *dataLogFormat* contains a token.

- `<setMetadataEvalTokens fieldName="" evaluateTokens="">`

  Here, the value of *evaluateTokens* is the desired value of the metadata field and it can contain tokens. Use <onSuccess> to trigger elements when setting the value is successful and <onError> when it is not.

- GET parameters: these are added to the URL of the experiment to set the value of a metadata field at the start of the experiment. Use the *postName* of the <field> element to refer to the metadata field:

  `frinexstaging.mpi.nl/name_of_experiment/?postName=some_value`

Using conditionals

Some elements make it possible to trigger other elements depending on whether a certain condition is met. For example:

```
<hasMetadataValue fieldName="numberOfErrors" matchingRegex="4">
    <conditionTrue>
       <triggerMatching listenerId="triggerEndExperiment"/>
    </conditionTrue>
    <conditionFalse>
       <nextStimulus repeatIncorrect="false">
    </conditionFalse>
 </hasMetadataValue>
```

A short explanation of some commonly used conditionals:

`<currentStimulusHasTag tags="">`
: As the element's name implies, it checks whether the current stimulus has one or more of the tags specified in *tags*. Multiple tags need to be separated by spaces.

`<hasMetadataValue fieldName="fieldName" matchingRegex="">`
: This checks whether the value of a metadata field matches the value in *matchingRegex*.

`<matchOnEvalTokens evaluateTokens="" matchingRegex="">`
: This checks whether the value of *evaluateTokens* matches the value in *matchingRegex*. Here, *evaluateTokens* can include tokens (see Appendix B).

`<stimulusHasResponse groupId="" matchingRegex="">`
: With this element you can check whether any response (if *groupId* is empty) has been given to the current stimulus, or (if *groupId* is given) whether a response has been given to a certain stimulus group. If so, it checks whether this matches the value in *matchingRegex*.

Each of these requires the <conditionTrue> and <conditionFalse> elements. Note that <matchOnEvalTokens> requires an additional <onError> element that triggers in case of an error in evaluating the value of *evaluateTokens.* If there are no elements in <conditionTrue> or <conditionFalse>, you can make them self-closing.

In combination with triggering blocks of elements conditionals can make creating complex setups more manageable.

9 | Creating questionnaires

For creating questionnaires you can of course use LimeSurvey or Qualtrics, but sometimes it makes sense to create a questionnaire within Frinex.

Using <withStimuli>

Instead of presenting trials one after the other using <loadStimulus> and <hasMoreStimulus>, you can present trials in vertical order on the page using <withStimuli>. This is useful for questionnaires, as it makes for a relatively simple structure of the presenter. Below you see the basic structure of <withStimuli>:

```
<withStimuli>
    <beforeStimulus>      Elements triggered once before the first trial (will be shown at the top)
    <eachStimulus>        Elements triggered for each trial
    <afterStimulus>       Elements triggered once after the last trial (will be shown at the bottom)
    <randomGrouping>      Works like in <loadStimulus>
    <stimuli>             Works like in <loadStimulus>
</withStimuli>
```

Stimulus rating buttons

If you use <withStimuli> to present your stimuli, then instead of <ratingButton>, <ratingRadioButton> and <ratingCheckbox> you need to use <stimulusRatingButton>, <stimulusRatingRadio> and <stimulusRatingCheckbox>. These take their answer options from the *ratingLabels* attribute in each trial's <stimulus> element.

Moreover, adding an ordinary rating button will produce a rating button for each stimulus, but they will not be linked to the separate stimuli in the way the stimulus rating buttons are. This means the responses to them will not be registered in the database.

To use withStimuli together with a stimulus rating button in a questionnaire, we need to add a question and a rating button for each stimulus to the basic structure. We also need to check the responses to make sure participants actually checked the button.

Example

```
<withStimuli randomise="false" maxStimuli="10">
    <beforeStimulus>
        <htmlText featureText="Question category XYZ"/>
    </beforeStimulus>

    <eachStimulus>
        <stimulusLabel/>
        <addPadding/>
        <stimulusRatingButton groupId="ratingResponse"/>
        <addStimulusCodeResponseValidation groupId="ratingResponse"
                                           validationRegex=".+"
                                           featureText="Please make a choice"/>
    </eachStimulus>

    <afterStimulus>
        <actionButton>
            <validateStimuliResponses>
                <conditionTrue>
                    <gotoNextPresenter/>
                </conditionTrue>
                <conditionFalse/>
            </validateStimuliResponses>
        </actionButton>
    </afterStimulus>

    <randomGrouping/>
    <stimuli/>
</withStimuli>
```

These group IDs need to match!

Tells Frinex that the stimulusRatingButton requires at least one character as a response

After pressing the button, this checks whether the response corresponds to *validationRegex.*

If it corresponds, go to the next screen / part of the experiment.

```
<stimuli>
    <stimulus identifier="1" label="Question 1 here" ratingLabels="1,2,3,4,5" tags="main"/>
    <stimulus identifier="2" label="Question 2 here" ratingLabels="Never,Sometimes,Always" tags="main"/>
    <stimulus identifier="3" label="Question 3 here" ratingLabels="Yes,No" tags="main"/>
    <stimulus identifier="4" label="Question 4 here" ratingLabels="Yes,No" tags="main"/>
</stimuli>
```

If different questions have different response requirements, then simply add more <addStimulusCodeResponseValidation> elements, each with a groupId that corresponds to the groupId of the intended question(s). See Appendix A for examples of regular expressions to use as input requirement in validationRegex.

The above method is most useful if many questions are of the same type. It is still useful if you have several types of questions and multiple questions per type. Then you could organize <eachStimulus> like this:

```
<eachStimulus>
    <stimulusLabel/>
    <addPadding/>

    <currentStimulusHasTag tags="Questions_123">
        <conditionTrue>
            <stimulusRatingRadio groupId="ratingResponse"/>
        </conditionTrue>
        <conditonFalse>
            <currentStimulusHasTag tags="Questions_456">
                <conditionTrue>
                    <stimulusRatingCheckBox groupId="ratingResponse"/>
                </conditionTrue>
                <conditonFalse>
                    <!-- all other questions -->
                    <stimulusRatingButton groupId="ratingResponse"/>
                </conditonFalse>
            </currentStimulusHasTag>
        </conditonFalse>
    </currentStimulusHasTag>

    <addStimulusCodeResponseValidation groupId="ratingResponse"
                                       validationRegex=".+"
                                       featureText="Please make a choice"/>
</eachStimulus>
```

Here, <currentStimulusHasTag> is used to distinguish between different types of questions (one uses radio buttons, another checkboxes and another push buttons). Make sure to add these distinguishing tags to the *tags* attribute of each <stimulus> element in the main <stimuli> section of your experiment file.

What if your questions are all of a different type? Then it is probably more practical to hard-code all questions using <ratingButton>, <ratingRadioButton> and <ratingCheckbox> than to use a lot of <currentStimulusHasTag> elements. If you have other stimuli besides those in your questionnaire in your XML, make sure to set the *maxStimuli* attribute of <withStimuli> to “1” and define one <stimulus> with a unique tag.

Tables and withStimuli

If you want every question to be presented as part of a table, the <table> element comes before the <withStimuli> element. Then, in <eachStimuli> you define the stimulus as for example a rating button within a row. For example:

```
<table>
    <withStimuli randomise="false" maxStimuli="10">
        <beforeStimulus/>

        <eachStimulus>
            <row>
                <column>
                    <stimulusLabel/>
                </column>
                <column>
                    <stimulusRatingButton/>
                </column>
            </row>
        </eachStimulus>

        <afterStimulus/>
        <randomGrouping/>
        <stimuli/>
    </withStimuli>
</table>
```

This will produce the stimulus label and some answer options to the right of it, for every stimulus.

Appendix A | Regular expressions

Below overview contains a small collection of common regular expressions that you can copy to your XML and adapt to your own needs. For a more complete overview and interactive testing of regexes, see for example regex101, RegExr or RegexPal (make sure to set the regex engine / flavor to Java or Javascript).

Note that when setting <field> elements in <metadata>, the `{` and `}` in a regex should both be surrounded by single quotes. So `{1,3}` becomes `'{'1,3'}'`. In all other places in the XML this is not necessary.

Regular expression	Matches with...
`"Psycholinguistics"`	The exact character string “Psycholinguistics” (case sensitive)
`"."`	Any single character
`"[mpi]"`	A single character of m, p or i
`"[^mpi]"`	Any single character except m, p or i
`"[0-9]"`	A single number between 0 and 9
`"[a-zA-Z]"`	A single character in the alphabetic range a to z or A to Z
`".*"`	Any character string of any length (including zero length)
`".+"`	Any character string of at least one character
`".{3,}"`	Any character string of at least three characters
`".{3,5}"`	Any character string of between 3 and 5 characters in length
`"a{3,5}"`	A character string consisting of only a’s between 3 and 5 characters in length
`"never|sometimes|always"`	Either of the exact strings “never”, “sometimes” or “always”
`"never|sometimes|"`	Either of the exact strings “never”, “sometimes” or “” (no value)
`"[\\s\\S]{1,}"`	Any string of 1 or more characters, multi-line input allowed (for multi-line input boxes)

Appendix B | Tokens

Tokens allow you to refer to the values of metadata fields. These metadata fields can be built into Frinex or they can be metadata fields you defined in the XML yourself.

The table below is an overview of the tokens in Frinex. Be aware that token names are case-sensitive, so for example ::stimuluscode:: does not refer to the *code* attribute of a <stimulus> element, but ::stimulusCode:: does.

token	Refers to	Extra information
::stimulusCode::	<stimulus> *code* attribute	
::stimulusCorrectResponses::	<stimulus> *correctResponses* attribute	
::stimulusId::	<stimulus> *identifier* attribute	
::stimulusLabel::	<stimulus> *label* attribute	
::stimulusPauseMs::	<stimulus> *pauseMs* attribute	
::stimulusRatingLabels::	<stimulus> *ratingLabels* attribute	Use ::stimulusRatingLabel_0:: for the first rating label, ::stimulusRatingLabel_1:: for the second rating label, and so on.
::stimulusTags::	<stimulus> *tags* attribute	
::stimulusResponse::	Response to the current stimulus	
::stimulusResponse_(stimulusId)_(groupId):: [3]	Response from a specific stimulus and/or button (including input boxes)	groupId corresponds to the *groupId* attribute of a button or input box element.
::completionCode::	Completion code	
::currentDateDDMMYYYY:: [4]	The current date in DDMMYYYY	
::userId::	The Id of the current participant	
::(listenerId of timer):: ::mediaLength_(mediaId)::	The current value of timer Last known length in seconds of media associated with 'mediaId'	
::metadataField_(postName of <field> element)::	Value of any metadata field defined in <metadata>	

[3] When using only *groupId*, the token becomes **::stimulusResponse__(groupId)::** (mind the double underscore)
When using only *stimulusId,* it becomes **::stimulusResponse_(stimulusId)_::** (mind the trailing underscore)
[4] Other date formats are allowed, for example DDMM, MMYYYY or YY.

Elements that accept tokens

If an element has an *evaluateTokens* attribute, you can use a token in that attribute. Some other elements have different attributes that also accept tokens:

Element	Usage
<stimulusFreeText>	Use tokens in *validationRegex*
<setMetadataValue>	Use tokens in *dataLogFormat*
<triggerMatching>	Use tokens in *listenerId*
<logTokenText>	Use tokens in *dataLogFormat*
<stimulusHasResponse>	Use tokens in *groupId*
<addStimulusCodeResponseValidation>	Use tokens in *groupId*
<setStimulusCodeResponse>	Use tokens in *groupId* and *codeFormat*

Some elements that don't accept tokens have a token-enabled counterpart:

No token	Token-enabled (link)	Usage
<htmlText>	<htmlTokenText>	Use tokens in the *featureText*
<actionButton>	<actionTokenButton>	Use tokens in the *featureText*
<stimulusImage>	<stimulusCodeImage> <stimulusCodeImageButton>	Use tokens in *codeFormat* to specify the media filename. Idem, but image is a button at the same time
<stimulusAudio>	<stimulusCodeAudio>	Use tokens in *codeFormat* to specify the media filename.
<stimulusVideo>	<stimulusCodeVideo>	Use tokens in *codeFormat* to specify the media filename.
<regionStyle>	<regionCodeStyle>	Use tokens in the *styleName*.
<pause>	<evaluatePause>	Use tokens in *evaluateTokens.* Also allows to specify minimum and maximum values. Requires <onSuccess> and <onError>.

Token methods

There are several token 'methods' that you can use within *evaluateTokens* attributes. They can be used in a similar way to functions in programming languages: the name followed by one or more arguments in parentheses. These arguments can contain tokens, which makes token methods very useful in complex setups. Within a method's parentheses, quotes should be either single straight quotes (`'`), or escaped double quotes (`&quot;`).

Token method
addTime(time1, time2)
daysBetween(date1, date2)
getRandomItem('a,b,c,d')
length(text)
random(number)
replaceAll(text, to_replace, replace_with)

See the XML Usage page for explanation and examples on these token methods.

Appendix C | Troubleshooting

This section is work-in-progress, but could be useful in the present state.

Error messages on the Frinex build page

Error messages about the contents of the XML file

⇨ These can be resolved by changing your XML file.

The 'XML Language Support' extension by Red Hat provides tips about the correct use of XML and the Frinex elements and attributes in Visual Studio Code / VSCodium. This works best if the *xsi:noNamespaceSchemaLocation* attribute of the main <experiment> element is set to 'http://frinexbuild.mpi.nl/**version**.xsd', where version can be 'stable' , 'beta' or 'alpha'.[5]

'The processing instruction target matching "[xX][mM][lL]" is not allowed.'

⇨ Make sure that the following is on the first (and not the second) line in the XML document:
`<?xml version="1.0" encoding="UTF-8" standalone="yes"?>`

'Open quote is expected for attribute "X" associated with an element type "Y".'

The XML should only contain straight quotes: ' or " (not ', ` or ")

⇨ You are likely to find the problematic quote in most editors by looking for a change in the syntax highlighting that starts at a certain location, continuing until the end of the XML file.

'404 Not Found'

' Whitelabel Error Page / This application has no explicit mapping for /error, so you are seeing this as a fallback.'

'WARNING: An illegal reflective access operation has occurred' (possibly including lines starting with ERROR)

The following cases can cause this error:

- <stimulusVideo> *showControls* or *autoPlay* attribute has empty value (`=""`)
- <stimulusAudio> *autoPlay* attribute has empty value
-

'The frinexVersion="beta" is too ambiguous for production deployments. Please specify the Frinex version for example frinexVersion="1.7.XXXX-stable". You can find the version number on the initial page of the staging version of your experiment FRINEX Version: 1.7.XXXX-stable.'

⇨ Rebuild your experiment to staging, click 'browse' in the staging column to see the version number

[5] Some older experiments use 'frinex.xsd', but this can give unsolvable errors in VSCode / VSCodium

Unexpected behavior on the Frinex build page

The building of your experiment seems stuck / the Frinex build page won't load

- ⇨ Contact Peter Withers (developer of Frinex)
  If he is not available within a reasonable time, you can ask Gert-Jan de Bresser or Tobias van Valkenhoef (system administrators)