JATOS Documentation

Last generated: October 24, 2018

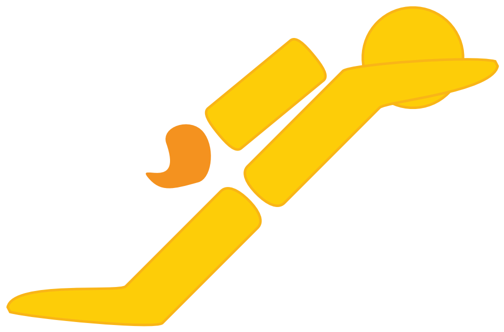


Table of Contents

What is JATOS?

JATOS (Just Another Tool for Online Studies) helps you set up and run your online studies on your own server.

This is the documentation for JATOS 3 - Are you looking for the docs of JATOS 2?

We have a page that covers all about migrating from JATOS 2 to 3 (page 12).

JATOS at a glance:

- Run studies on your **own server**. This means that you keep complete control over who can access your result data and can comply with your ethics.
- Studies run on **mobile phones, tablets, desktops, and lab computers** - any device with a browser.
- Use tools like **jsPsych** or **lab.js** to prepare your study - or write all **HTML / JavaScript / CSS** yourself and have full control.
- **Run group studies** where multiple workers interact with each other in real-time.
- It's GUI-based, so there's no need to use the terminal to talk to your server.
- Recruit participants via **Amazon Mechanical Turk**, **Prolific** etc.
- It's **open-source and free** to use.
- **Manage workers**, to e.g. make sure that each participant does your study only **once**.
- **Export/Import** studies to facilitate exchange with other researchers.

You can try out JATOS on *cortex*, our test server (page 18).

The whole documentation is available as PDF Download

Please cite us if you use JATOS for your research.

JATOS is open-source and released under the Apache 2 Licence. The source code is available on github.com/JATOS/JATOS.

Download the latest release

Contact us

JATOS is under active development, so please do get in touch to ask questions, suggest enhancements and report bugs. We really appreciate any contributions.

We also conduct workshops: If you want us to give a lecture or workshop about JATOS and/or online studies in general contact us via email.

GitHub issues

If you would like to report a bug or suggest a new feature that could improve JATOS, you could write a GitHub issue.

Stack Overflow / Google group

If you have a question about JATOS, write us (and the user community) an email through our Google Group or ask it at stackoverflow.com.

Email

If you have a question about JATOS or need help with your experiments, write to either:

elisa.filevich@gmail.com

lange.kristian@gmail.com

Installation

Easy installation on your local computer

JATOS runs on MacOS X, MS Windows and Linux

A local installation is straightforward.

Usually you first develop your study with JATOS on a local computer. Then in a second step you bring it to a server installation of JATOS.

With a local installation only you have access to JATOS - with a server installation (page 74) others can run your study via the internet too. This is especially true if you want to publish your study on Mechanical Turk.

For convenience JATOS is available in a variant bundled with Java.

To run JATOS, you need Java installed on your computer (to be precise, you need a Java Runtime Environment, aka JRE). Chances are, you already have Java installed. To check whether Java is installed on your system, type `java -version` in your terminal (MacOS X / Linux) or command window (MS Windows). If you don't have Java installed, you can either download and install it directly or download and install JATOS bundled with Java, according to your operating system.

Installation MS Windows

1. Download the latest JATOS release (exchange 'xxx' with the current version)
    - Without Java: *jatos-xxx.zip*
    - Bundled with Java: *jatos-xxx_win_java.zip*
2. Unzip the downloaded file. You can place the unzipped folder pretty much anywhere, **except** in a folder that synchs across devices, like Dropbox or Google Drive. Find out (page 0) more about why not.
3. In the File Explorer move to the unzipped JATOS folder and double-click on `loader.bat`. (Or `loader` alone, if your filename extensions are hidden). A command window will open and run your local JATOS installation. Simply close this window if you want to stop JATOS.
4. All set! Now go to the browser of your choice and open http://localhost:9000/jatos/login. You should see the login screen (wait a moment and reload the page if you don't). Login with username 'admin' and password 'admin'.

Installation MacOS X and Linux

1. Download the latest JATOS release (exchange 'xxx' with the current version)
    - Without Java: *jatos-xxx.zip*
    - For MacOS bundled with Java: *jatos-xxx_mac_java.zip*
    - For Linux bundled with Java: *jatos-xxx_linux_java.zip*
2. Unzip the downloaded file. You can place the unzipped folder pretty much anywhere, **except** in a folder that synchs across devices, like Dropbox or Google Drive. Find out (page 0) more about why not.
3. In your terminal window, cd into the unzipped JATOS folder
4. Run the loader shell script with the command `./loader.sh start` (You might have to change the file's permissions with the command `chmod u+x loader.sh` to make it executable). Ignore pop-ups like 'To use the java command-line tool you need to install a JDK' - just press 'OK'.
5. All set! Now go to the browser of your choice and open http://localhost:9000/jatos/login. You should see the login screen (wait a moment and reload the page if you don't). Login with username 'admin' and password 'admin'.

Your local JATOS installation will run in the background. If you want to stop it, just type `./loader.sh stop` in your terminal window.

How to go on from here

The easiest way to start with JATOS is to download and import one of the example studies (page 0) and play around with it (page 7).

Get started

We've made an video: Introduction into JATOS

Get started in 4 steps

1. **Download JATOS and install a local instance (page 5)**
2. **Open JATOS' GUI by going to http://localhost:9000/jatos/login in your browser window**
3. **Download and import an example study**
    a. Download one of the Example Studies, e.g. the 'Go- / No-Go Task' with jsPsych. Do not unzip the downloaded file.
    b. Import the study into JATOS: Go to JATOS' GUI in your browser and click on **Import Study** in the header. Choose the .zip file you just downloaded. The imported study should appear in the sidebar on the left.
4. **Explore the GUI**

In the sidebar click the study to get into the study's page.

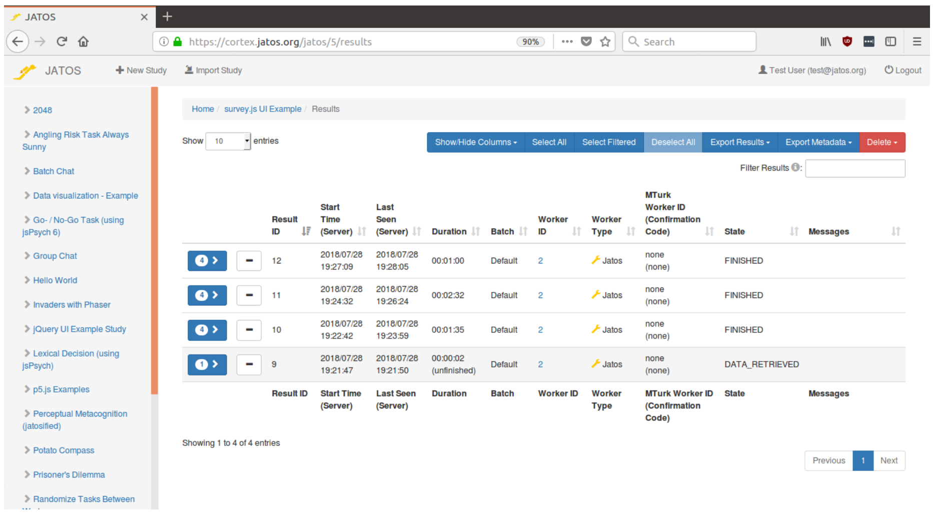
To do a test run of the entire study, click on **Run** in the toolbar on top of the page.

If you finished running through the study, you can check the results.

- To see whole-study results, click on the **Results** button on the top of the page.
- To see results from individual components, click on the **Results** buttons on each component's row.

For example, you can see each result's details by clicking on the little arrow to the left of its row (more information on how to mangage results (page 0)).

Here's a screenshot of a study's results view:



https://cortex.jatos.org/jatos/5/results

Home / survey.js UI Example / Results

Show 10 entries

Filter Results:

		Result ID	Start Time (Server)	Last Seen (Server)	Duration	Batch	Worker ID	Worker Type	MTurk Worker ID (Confirmation Code)	State	Messages
4	−	12	2018/07/28 19:27:09	2018/07/28 19:28:05	00:01:00	Default	2	Jatos	none (none)	FINISHED	
4	−	11	2018/07/28 19:24:32	2018/07/28 19:26:24	00:02:32	Default	2	Jatos	none (none)	FINISHED	
4	−	10	2018/07/28 19:22:42	2018/07/28 19:23:59	00:01:35	Default	2	Jatos	none (none)	FINISHED	
1	−	9	2018/07/28 19:21:47	2018/07/28 19:21:50	00:00:02 (unfinished)	Default	2	Jatos	none (none)	DATA_RETRIEVED	

Showing 1 to 4 of 4 entries

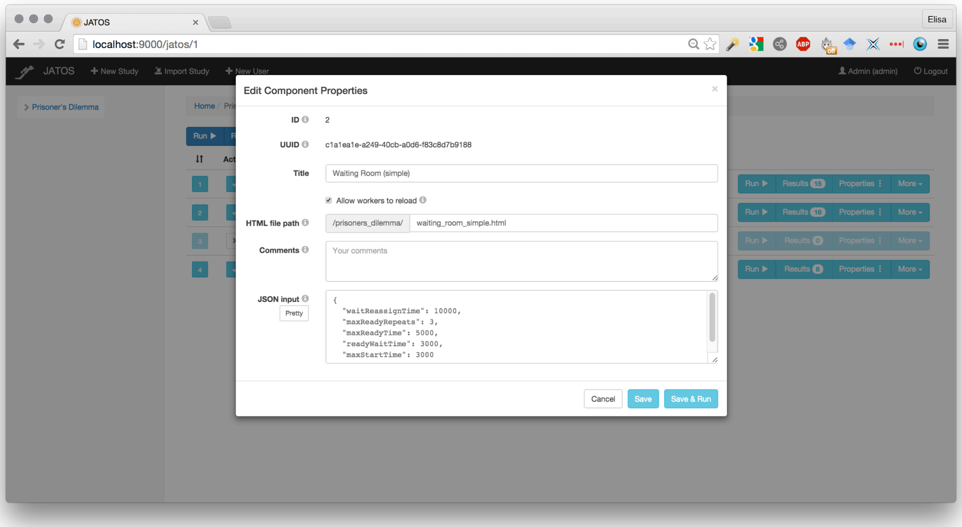
Explore

Now it's time to explore a little bit more.

- You can click on any component's position button and drag it to a new position within the study.
- Each component has a **Properties** button. The component's HTML file may read the data in the field 'JSON data'. This is a way to make changes in the details of the code (wording of instructions, stimuli, timing, number of trials, etc) without having to hard-code them into JavaScript.
- Where are the actual HTML, JavaScript, and CSS files? They are the files that actually run your study, so make sure you can locate them. All these files, together with any images, sound files, etc. you might have, are called "Study assets". They will be in `/path_to_my_JATOS/`

`study_assets_root/name_of_my_study/`.

Here's a screenshot of a component's properties view:



Edit Component Properties

ID: 2

UUID: c1a1ea1e-a249-40cb-a0d6-f83c8d7b9188

Title: Waiting Room (simple)

☑ Allow workers to reload

HTML file path: /prisoners_dilemma/ waiting_room_simple.html

Comments: Your comments

JSON input

```
{
  "waitReassignTime": 10000,
  "maxReadyRepeats": 3,
  "maxReadyTime": 5000,
  "readyWaitTime": 3000,
  "maxStartTime": 3000
```

Set up your own studies

So, now you've installed JATOS and tried out all the examples studies. What now?

Create a new study

There are two ways to create a new study:

1. from scratch by pressing the **New Study** button in the header of each JATOS page. Then edit the study properties and add new components manually.
2. take an existing study (e.g. from Example Studies (page 0)) as a prototype and modify it bit by bit. We recommend you start with this to get a quick idea of how JATOS works. Just import an existing study (e.g. one from the study examples) and clone it by clicking on More –> Clone in the study bar.

Write your own studies in JavaScript

The most difficult part - though it's still easy! - is to learn to write your own study component scripts, using HTML, CSS and JavaScripts. Or, instead of reinventing the wheel, you could use a framework like jsPsych that helps you with this (see jsPsych and JATOS (page 25)).

Check out the Mandatory lines in your components' HTML (page 19) page to know what you absolutely must include in your scripts in order to let JATOS see them. Also check out the jatos.js Reference (page 100), that contains a set of very useful functions that you have to use to communicate with jatos (to e.g. submit and receive data).

If you are a newbie to HTML/JavaScript programming, there are LOADS of free and excellent tutorials online. Like this one from the Kahn Academy or more searchable tutorials like the simple ones from the w3 schools. In addition, StackOverflow is the best place to solve the problems that hundreds of others have encountered before.

Import / Export of studies

Usually you conveniently develop your study on your local computer where you have a local installation of JATOS (page 5). Then just use the export and import buttons in your installations to transfer the study to your JATOS server (page 74).

1. In the GUI of your local installation press **Export** in the Study Toolbar. JATOS saves your study asset folder and some data about your study and it's components (mostly the properties) into a ZIP file and lets you

download it. Leave the ZIP file as it is.

2. In the GUI of your server installations press **Import** in the header. Select the ZIP you saved in step 1. JATOS will upload and unpack your study.

If you have trouble with the export and you are using a Safari browser have a look into this issue in our Troubleshooting section (page 62).

Decide how you're going to recruit your workers and generate the links

Once you have your study running and have it on a server instance, you'll need to get your workers to access your study. Different types of workers (page 38) might be allowed to run your study. You could, but you don't have to, recruit your workers using MTurk (page 43). You could also generate direct links in the Worker Setup (page 0) to send to different workers.

Export your result data

After you let workers run your study and you gathered result data you probably want to export them to your local computer for further analysis. Go to one of the **Results** views and select the results you want to export (select them by just clicking somewhere in the row). Then click 'Export Selected'. This will download all your results in one text file.

Analyse your data

Once you have collected all your data, export it to text files. If you used the JSON format (which is handy - but any other text is fine too) you can analyse your data using this nice JSON parser for Matlab and Octave or the JSON parser for R.

Migrate from JATOS 2 to JATOS 3

Updating from JATOS 2 to JATOS 3 in either your local or server installation is mostly straightforward: most studies you wrote for JATOS 2 will work for JATOS 3 out of the box. (Although we added some nice new features like the Batch Session or the user manager). The only exception are studies that used the Group Session: you'll have to change a couple of functions to access the Group Session.

Update your studies - changes in jatos.js

If you used the **Group Session**, you'll have to change a couple of lines in your JavaScript. To get your JATOS 2 code running in JATOS 3, simply change:

- Writing to the Group Session: `jatos.groupSessionData = myObject;` -> `jatos.groupSession.setAll(myObject);`
- Reading from the Group Session: `var myObject = jatos.groupSessionData;` -> `jatos.groupSession.getAll();`

But consider going beyond this quick fix: the new Group Session offers much finer access with other functions, not just the `setAll` and `getAll` (which is why we changed them, not *just* to annoy you!). Additionally, we recommend to use the callback / promise functions in case something fails.

To improve the Group Session, we abolished the direct access to the Group Session data with `jatos.groupSessionData`. Now you can only access it via several functions with the form: `jatos.groupSession.[functionName]`, e.g. `jatos.groupSession.get()` and `jatos.groupSession.set()`. Find details about the jatos.js functions to access the Group Session in the jatos.js Reference / Functions to access the Group Session (page 132) and examples of how to use them in Write Your Own Group Studies (page 0).

Also, consider that JATOS 3 includes the **Batch Session** and things (like randomize conditions between participants (page 0)) that you previously did with the Group Session can now be done easier with the Batch Session. Handling the Batch Session doesn't involve joining a group first - the batch channel is opened automatically by jatos.js for you. Check out the differences between the group- and Batch Sessions in Session Data - Three Types (page 28) to decide which session type is best for your needs.

Update your local JATOS installation

Just follow Update JATOS (page 60) - remember to stop JATOS and to make backups before doing anything.

Update your server JATOS installation

Follow Updating a JATOS server installation (page 95).

With JATOS 3 now requires (it's no longer optional) support for WebSockets (otherwise the Batch and Group Session wouldn't work). So if you use a proxy in front of JATOS it has to be configured accordingly. You can find examples for Apache (page 93) and Nginx (page 85) in this documentation.

Keep your database with the update

The schema of the database changed from JATOS 2 to JATOS 3. JATOS does this automatically. But keep in mind that this is irreversible: once JATOS updated the database it can't be used in JATOS 2 anymore. So always back up your database in a safe place before upgrading, in case anything goes wrong.

Data Privacy and Ethics

What does JATOS store?

Data privacy is a critical issue in online studies. You should be careful when collecting, storing and handling data, regardless of which platform you use to run your studies.

We developed JATOS with data privacy in mind, preventing any breaches of the standard ethical principles in research. However, ultimately you are responsible for the data you collect and what you do with it.

DATA SECURITY
IN THIS CORNER, WE HAVE FIREWALLS, ENCRYPTION, ANTIVIRUS SOFTWARE, ETC. AND IN THIS CORNER, WE HAVE DAVE!!
HUMAN ERROR

(copyright 2006 John klossner, www.jklossner.com)

Here are a few advantages and limitations of JATOS with regards to data privacy. Please read them carefully before you run any study, and please contact us (page 4) if you find that these are not sufficient, or have suggestions for improvement.

- JATOS' main advantage is that you can store your participants' data in your own server, and not in a commercial server like Amazon or Qualtrics. This means that you have full control over the data stored in your database, and no commercial company has access to it. JATOS does not share any data (except of course during a study run with the participant's browsers). Each JATOS installation is completely independent of any other JATOS installation.
- By default, JATOS stores the following data:
    - time (of the server running JATOS) at which the study -and each of its components- was started and finished

    - the worker type (page 38) (MTurk, General single, Personal multiple, etc)
    - in cases of MTurk workers, the confirmation code AND the MTurk worker ID. In these cases, if an MTurk worker participated in two of your studies, running in the same JATOS instance, **you will be able to associate the data across these two studies**. This is an important issue: MTurk workers might not be aware that you are the same researcher, and will not know that you have the chance to associate data from different studies. The best way to avoid this is to export all your study's data and delete it from the JATOS database once you are done with it. In this way, JATOS won't know that a worker already participated in another study and will create a new worker ID for them.
- JATOS will **not** store information like IP address or browser type (or any other HTTP header field).

Things you should consider in your studies

- You should consider to add some button in your study pages to abort the study. Some ethics demand that any participant should have the **right to withdraw** at any time, without explanation. In this case all data of the participant gathered during the study should be deleted. Conveniently jatos.js offers an abortStudy method (page 110) that does exactly that.
- Use **encryption** with your server instance (page 74). Only with encryption no one else in the internet can read the private data from your study's participants.
- JATOS will **not** store information like IP address or browser type (nor any other HTTP header field). However, you could access and store this information through your JavaScripts. You could also record whether workers are actively on the browser tab running your study, or whether they have left it to go to another tab, window or program. If you collect any of these data, you should let your workers know.
- Bear in mind: Every file within your study assets folders is public to the Internet. Anybody can in principle read any file in this folder, regardless of how secure your server is. **Thus, you should never store any private data, such as participants' details in the study assets folders.**
- **Do not store private information in the Batch Session or Group Session.** Both sessions are shared between all members of a batch or group respectively. If you store private data any other member of this batch or group could potentially access it. Since the Study Session is only shared within the same study run it is not a problem to store private information there.

Cookies used by JATOS

Sometimes it is neccessary to specify which cookies are stored in a participants browser. JATOS knows three types of cookies and only two of them are stored in a participants browser.

1. Up to ten JATOS ID cookies with cookie name JATOS_IDS_* (can be a number from 0 to 9)*

These cookies store values about each study run. JATOS allows up to 10 study runs in parallel per browser - therefore there are up to 10 JATOS ID cookies.

All IDs are used only by JATOS internally and do not allow the identification of the worker.

The cookie virtually never expires (actually far in the future, around the year 2086).

HttpOnly is set to true (this means, it can't be read by JavaScript within the browser).

This cookie contains these parameters:

- *studyId*: identifier of the study
- *batchId*: identifier of the batch
- *componentId*: identifier of the component
- *componentPos*: position of the component within the study
- *workerId*: identifier of the worker used internally to identify the worker anonymously
- *workerType*: there are 5 worker types with different use cases in JATOS
- *componentResultId*: identifier of the component result (a component result is used to store data of the component run)
- *studyResultId*: identifier of the study result (a study result is used to store data of this study run)
- *groupResultId*: identifier of the group this worker belongs to (null if it isn't a group study)
- *creationTime*: timestamp (epoch time) of this cookie's creation
- *studyAssets*: name of the directory where the study's assets are stored on the JATOS server
- *jatosRun*: State of a study run with a JatosWorker. If this run doesn't

belong to a JatosWorker this field is null. It's mainly used to distinguish between a full study run and just a component run.

E.g. `JATOS_IDS_0:"batchId=108&componentId=306&componentPos=2&componentResultId=3867&creationT`

2. Cookie JATOS_GENERALSINGLE_UUIDS

This cookie is used by JATOS to store which study runs with a General Single worker already happened in this browser. It only stores a list of IDs that universally identifies a study (UUID).

3. Play Framework session cookie named PLAY_SESSION

This cookie is used only by JATOS' GUI and provides session and user info. It is **not** set during a study run and therefore does **not** store any worker related information.

The cookie's expires header field is set to Session, which mean that after the browser is closed the cookie will be deleted.

HttpOnly is set to true (this means, it can't be read by JavaScript within the browser).

This cookie contains the parameters:

- *userEmail*: username of the logged-in user (an email)
- *sessionID*: Play's session ID
- *loginTime*: user's login time in the GUI as a timestamp
- *lastActivityTime*: user's last activity time in the GUI as a timestamp

Additionally Play stores a hash of the whole cookie's data to check integrity of the cookie's data.

E.g. `PLAY_SESSION:"b6c01f2fa796603491aaed94168651b54b154ca1-userEmail=admin&sessionID=4k1atg9`

JATOS Tryout Server

Cortex is a running JATOS server where you can try out JATOS in the Internet instead of your local computer:

https://cortex.jatos.org (log in with *test@jatos.org* and *abc123*)

This is a normal JATOS installation with many example studies already imported. You can run the examples or import your own studies if you want to test them in a JATOS running in an online environment.

But be aware that every day **at 8:00 UTC this JATOS server will be reset** to its inital state and all changes will be lost. In other words this JATOS is not meant to run your studies online.

Mandatory lines in your components' HTML

The best way to write an HTML/JavaScript file that runs with JATOS is to use one of the HTML files from the Example Studies (page 0) as a starting point.

Here are the absolute basics that any component HTML file must have in order to correctly run with JATOS

1. A link to the jatos.js library in the head section

   ```
   <html>
     <head>
       <script src="/assets/javascripts/jatos.js"></script>
     </head>
   </html>
   ```

   Remember (page 63): Any URL or file path in a HTML file should only use `/` as a file path separator - even on Windows systems.

2. The second bit is not really necessary but without defining the `jatos.onLoad` callback function you won't be able to use jatos.js' features. Of course you could start right away with any JavaScript but if you want to use jatos.js' variables and functions you have to wait untill it is finished initializing. This initialization includes among others setting up the study and component JSON input variables and the study, batch, and group sessions. If you try to access one of jatos.js variables or functions before the initialization is finished it might lead to an error or wrong results.

   ```
   <script>
     jatos.onLoad(function() {
       // Start here with your code that uses jatos.js' var
   iables and functions
     });
   </script>
   ```

Adapt Pre written Code to run it in JATOS (Jatosify)

Make Your Existing Code Run in JATOS - or How To Jatosify a Study

You might have a task, experiment, survey, or study running in a browser. You might have all its files like HTML, JavaScripts, images, etc. Maybe you wrote it with jsPsych or got it from The Experiment Factory. And now you want to run it with JATOS? Then follow this page.

Create the study in JATOS

1. Create a new study with the '**New Study**' button in JATOS' header. Choose a study title and a folder name. Leave the other fields empty for now and click 'Create'. JATOS will have created a new folder within your assets root folder (default is `/path_to_your_JATOS/study_assets_root/`).
2. Copy all your files (HTML, JavaScripts, images, audio, …) into your new study folder.
3. Back in the JATOS GUI, and within the newly created study, create a **new component** by clicking 'Components' and then 'New'. Choose a component title and set the HTML file name, to the name of the HTML file you just copied into the study folder.
4. In your HTML, CSS and JavaScripts, for your paths you can choose between 1) relative paths or 2) absolute paths. Relative paths are usually shorter and easier to handle but are only available since JATOS version 3.2.3.
    a. **Relative paths (since v3.2.3, recommended))** Just use the relative path within your study's folder.
        - E.g. a file in your local filesystem `/path_to_your_JATOS/study_assets_root/group_snake/snake.css` turns to just `snake.css`
        - E.g. in a subfolder `/path_to_your_JATOS/study_assets_root/group_snake/subfolder/snake.css` turns to `subfolder/snake.css`

        If you want to reference a file in a higher folder use `../` in front (like in a normal filesystem).

        - E.g. your referencing file is `subfolder/myscript.js` and you want to reference `css/snake.css` use

`../css/snake.css`.

b. **Absolute paths)** Always use the prefix `/study_assets/` and then the study assets name you specified in your study's properties when you created it.

  - E.g. if you load the CSS file `snake.css` and the study's assets name is `group_snake` use `<link rel="stylesheet" type="text/css" href="/study_assets/group_snake/snake.css" />`
  - Or if you want to load some JavaScript from your local study assets with the name `prisoner_dilemma`, e.g. the jQuery library, use `<script src="/study_assets/prisoner_dilemma/jquery-1.11.1.min.js"></script>`

✰ For absolute paths make sure you understand the difference between the `study_assets_root` folder and the placeholder `study_assets` in your path names. `study_assets_root` is the folder in your system (or in the server) where the assets (HTML, JS, CSS, images, etc) of **all** your JATOS studies will be stored. You can configure (page 81) the location of this folder. `study_assets`, on the other hand, is just a placeholder that will go in your HTML files. JATOS will interpret this and replace the placeholder with the path, (specific to the study) that you entered in the field 'Study assets directory name' in your Study's Properties. The advantage of this is that you can change the location or name of the assets for any study, or export-import a study into a different computer, and the study will still run without having to make any changes in the HTML code.

5. Now it's time for a first glimpse: Click the '**Run**' button in either the study's or the component's toolbar. Your experiment should run like it did before without JATOS. Use the browser's developer tools to check for eventually missing files and other occurring errors.

Edit your HTML and JavaScript

Up to this point JATOS served as a mere provider of your files. Now we want to use a feature of JATOS: We want to store your result data in JATOS' safe database.

1. Include the **jatos.js** library in your HTML `<head>`

  - JATOS < v3.3.1) Add the line `<script src="/assets/javascripts/jatos.js"></script>`

- JATOS >= v3.3.1) Add the line `<script src="jatos.js"></script>`

2. Add **`jatos.onload`**

   Every study in JATOS starts with this call. So whatever you want to do in your study it should start there.

```
jatos.onLoad(function() {
  // initialize and start your JavaScript here
});
```

   E.g. if you want to initialize a jsPsych experiment:

```
jatos.onLoad(function() {
  jsPsych.init( {
    ...
  });
});
```

3. Now to actually send our result data to JATOS we use jatos.js' function **`jatos.submitResultData`**. We can pass this function any data in text format including JSON, CSV or XML.

   E.g. if we want to send a JavaScript object as JSON

```
var resultJson = JSON.stringify(myObject);
jatos.submitResultData(resultJson, jatos.startNextCompon
ent);
```

   Conveniently but optionally `jatos.submitResultData` takes a second parameter which specifies what should be done after the result data got sent. Usually one want to jump to the next component (`jatos.startNextComponent`) or finish the study (`jatos.endStudy`).

   Another example where we use jsPsych: We have to put `jatos.submitResultData` into jsPsych's `on_finish`:

```
jsPsych.init( {
  ...
  on_finish: function(data) {
    // Submit results to JATOS
    var resultJson = JSON.stringify(jsPsych.data.getDat
a());
    jatos.submitResultData(resultJson, jatos.startNextCo
mponent);
  }
});
```

That’s about it. Infos about other jatos.js functions and variables you can find in the reference (page 100).

Beyond the basics

- Think about dividing your study into **several components**. You could have separate components e.g. for introduction, training, experiment and feedback. You could even consider splitting the experiment into several parts. One advantage is that if your participant stops in the middle of your study you still have the result data of the first components. Also, you can re-use components in different studies.
- Use the study’s and component’s ‘**JSON input data**’. With them you can change variables of your code directly through JATOS’ GUI, which might come handy if someone isn’t good in JavaScript.
- You can add a **quit button** to your study to allow the participant to abort at any time (page 15).

lab.js and JATOS

lab.js is an easy to use tool to create online experiments. Their Builder makes creating an online experiment a piece of cake - although you can also write code yourself: lab.js supports this too.

lab.js and JATOS fit perfectly together: **lab.js directly exports JATOS studies**. So you don’t need to write or modify any bits of code. You can create your experiment with lab.js. Then just import your studies into JATOS and let particpants run it.

lab.js already has a great documentation and one page there is solely dedicated to JATOS: Collecting data with JATOS.

That’s all there is to say.

jsPsych and JATOS

JATOS basically cares for the server side: it stores result data, does worker management etc. JATOS doesn't care so much for what happens in the browser itself - your HTML, JavaScript and CSS. Of course you can write this all yourself, but you could also use a framework for this. A very good one is jsPsych.

In our example studies (page 0) are a couple of jsPsych ones.

Here are the necessary changes if you want to adapt your jsPsych experiment so that it runs within (and sends the result data to) JATOS. Additionally you can have a look at Adapt Pre written Code to run it in JATOS (Jatosify) (page 20).

How to turn your jsPsych experiment into a JATOS study

1. Include the `jatos.js` library in the `<head>`

   ```
   <script src="/assets/javascripts/jatos.js"></script>
   ```

   Remember (page 63): Any URL or file path in a HTML file should only use '/' as a file path separator - even on Windows systems.

2. Wrap jsPsych's init call `jsPsych.init` in a `jatos.onload` call

   ```
   jatos.onload(function() {
     jsPsych.init( {
       // ...
     });
   });
   ```

That's all. If you additionally want to send your result data to JATOS read on.

Send jsPsych's result data back to JATOS

Here we use jsPsych's function `jsPsych.data.getData()` (jsPsych 5) or `jsPsych.data.get().json()` (jsPsych 6) to collect the data into a JSON-formatted string. Then we use JATOS' function `jatos.submitResultData` to send your result to JATOS and ask JATOS to move to the next component, if there is one.

jsPsych 5

```
jatos.onload(function() {
  jsPsych.init( {
    // ...
    on_finish: function() {
      var resultJson = JSON.stringify(jsPsych.data.getData());
      jatos.submitResultData(resultJson, jatos.startNextCompone
nt);
    }
  }
});
```

jsPsych 6

```
jatos.onload(function() {
  jsPsych.init( {
    // ...
    on_finish: function() {
      var resultJson = jsPsych.data.get().json();
      jatos.submitResultData(resultJson, jatos.startNextCompone
nt);
    }
  });
});
```

Adding additional HTML snippets to a jPsych code (e.g. cancel button)

jsPsych has the habit of cleaning the HTML's body and fill it with its own code. This means that whatever you write between the `<body>` tags will be ignored. But you might want to add some additional HTML element like a cancel button to the page without changing the jsPsych plugin or writing a new one. How can this be done?

Luckily jsPsych offers a callback function on_load. Whatever we write in there is called after jsPsych did its body clean-up. So you could add your extra HTML elements in there.

Here's an example (you need jQuery for this one to work):

```
var my_trial = {
  type: 'some-plugin',
  on_load: function() {
    $("body").append('<button onclick="jatos.abortStudy()">Canc
el Study</button>');
  },
  ...
```

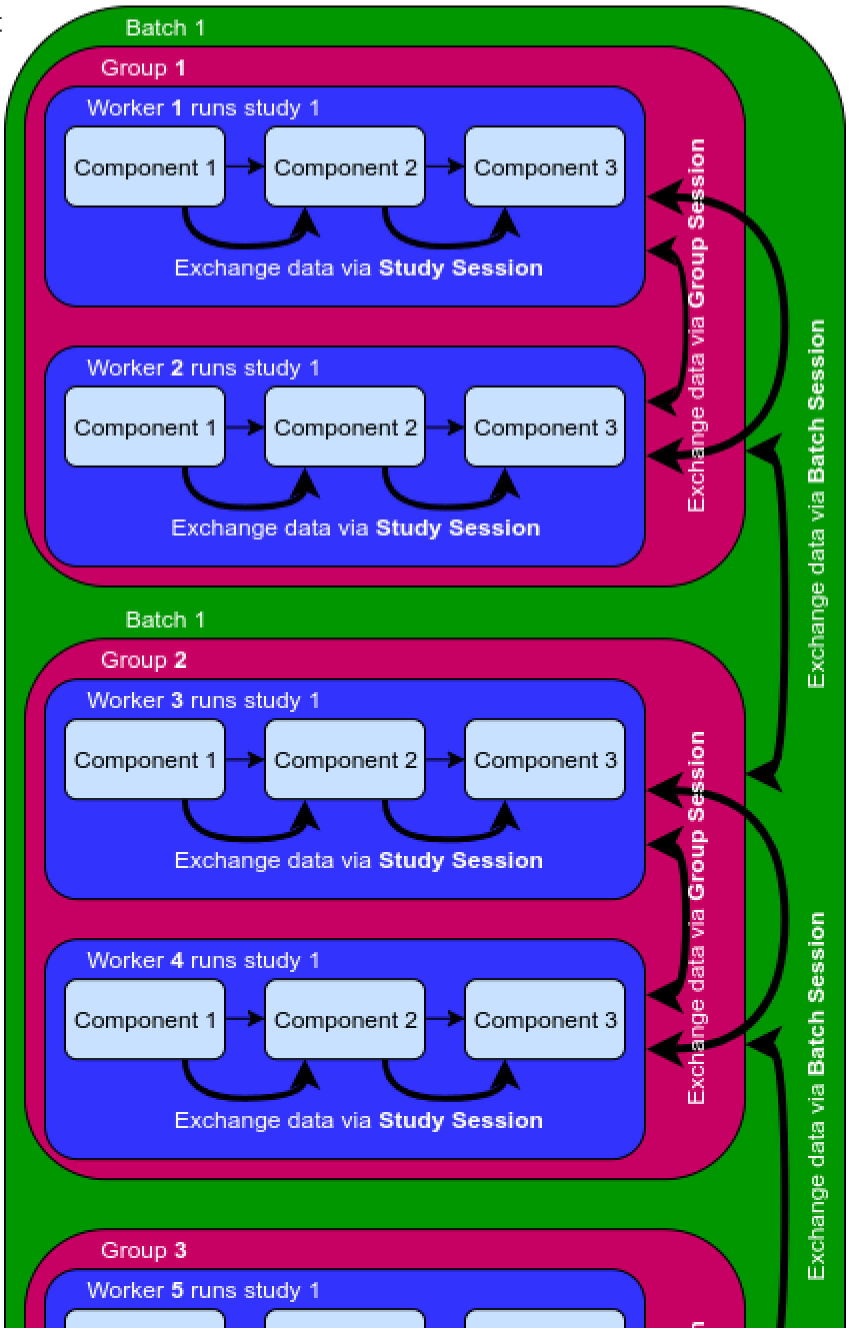
And without jQuery it's more cumbersome:

```
var my_trial = {
  type: 'some-plugin',
  on_load: function() {
    var button = document.createElement("button");
    button.innerHTML = "Cancel Study";
    document.body.appendChild(button);
    button.addEventListener("click", function() {
      jatos.abortStudy();
    });
  },
  ...
```

You probably want to add some styling but this is how it works in principle.

Session Data - Three Types

When to use?



Often you want to store information during a study run and share it with other components of the same study, or between workers of a group or batch. The three different session types let you transfer data in this way (shown by the curved arrows in the picture on the right). Workers can write into the sessions through jatos.js.

The data stored in the sessions are **volatile** - do not use the sessions to store data permanently. Instead, store any information that might be useful for data analysis in the **Result Data**. Unlike the data stored in the sessions, the Result Data are stored permanently in the JATOS server, and will never be deleted automatically.

The data stored in the sessions are not exported or imported together with a study.
If you want data to be exported with a study, use the **JSON Input Data** instead.

Comparative Overview

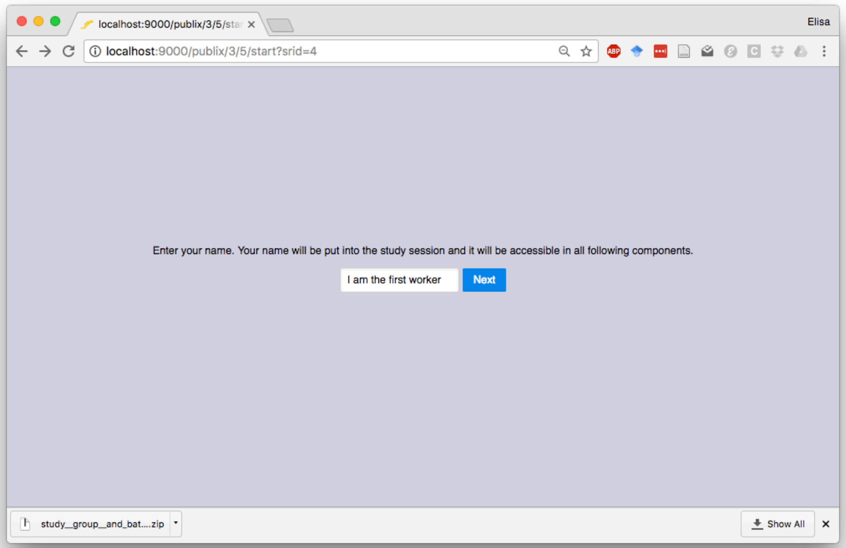
	Batch Session	Group Session	Study Session
Scope (accesible by)	All workers in a batch	All workers in a group	All components in a study
Usage	Exchange and store data relevant for all members of a batch	Exchange and temporarily store data relevant for all members of a group	Exchange and temporarily store data between components of a single study run
Example use	(Pseudo-)randomly assign conditions to different workers; Combine results from different groups working in the same batch	Store choices of the two members of a Prisoner's Dilemma game	Pass on correct answers between components; Keep track of the number of iterations of a given component that is repeated
Lifetime	Survives after all workers finished their studies	Automatically deleted once the group is finished	Deleted once the worker finished the study - Hence temporary
Updated when and via	Any time you call one of the jatos.batchSession functions (page 115)	Any time you call one of the jatos.groupSession functions (page 132)	At the end of each component or if you call jatos.setStudySessionData (page 0)
Visible and editable from JATOS' GUI	yes	no	no
Requires WebSockets	yes	yes	no

	Batch Session	Group Session	Study Session
Included in exported studies	no	no	no

Example Study

We have an example study (page 0), where we show the three different session types in action. Try it yourself:

1. Download and import the study. You'll find that the study contains two components: "First" and "Second".
2. Run the study once: easiest is as a JATOS worker (just click 'Run' on the study bar, not on any of the component bars).
3. The first component will prompt you for a name. It will then write the name you enter here into the **Study Session**. Because all components have access to the Study Session, the second component can read it and use your name in a chat window.

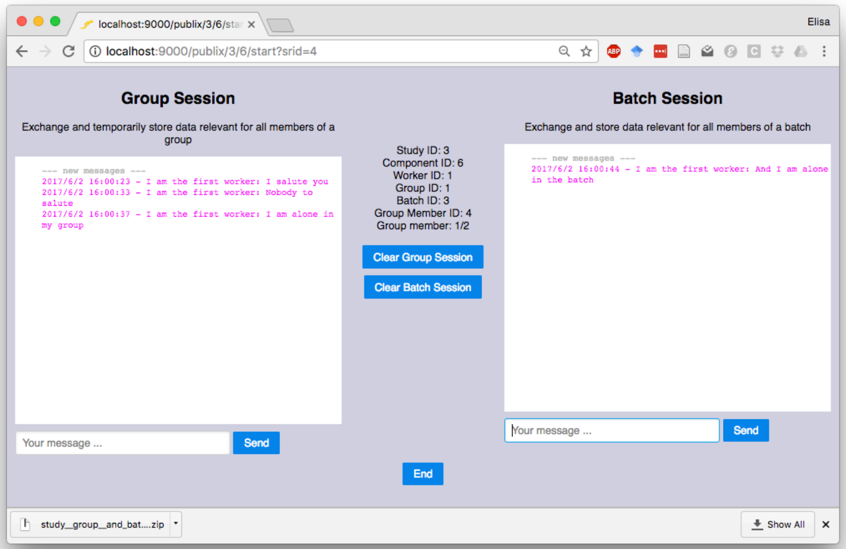

localhost:9000/publix/3/5/start?srid=4
Elisa
study_group_and_bat....zip
Show All

Enter your name. Your name will be put into the study session and it will be accessible in all following components.

I am the first worker | Next

4. When you click on 'Next', the second component will load. Here you will see two chat windows: The left one is called the group chat because it uses the Group Session; the right one is called batch chat because it uses the Batch Session. For now you're alone in these chat rooms. So, without closing this run and from new browser tabs, **run the study 2 more times**

(at least). You can choose any worker type you want. Additional runs with the JATOS worker will work but you can also use other worker types (page 0).

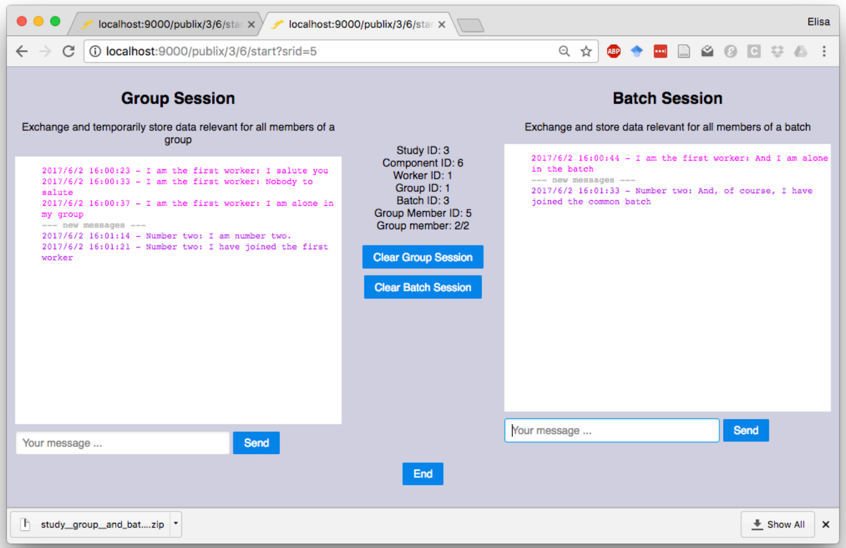

localhost:9000/publix/3/6/start?srid=4
Elisa
Group Session
Exchange and temporarily store data relevant for all members of a group
Batch Session
Exchange and store data relevant for all members of a batch
Clear Group Session
Clear Batch Session
Your message ...
Send
End
study_group_and_bat....zip
Show All

```
--- new messages ---
2017/6/2 16:00:23 - I am the first worker: I salute you
2017/6/2 16:00:33 - I am the first worker: Nobody to salute
2017/6/2 16:00:37 - I am the first worker: I am alone in my group
```

Study ID: 3
Component ID: 6
Worker ID: 1
Group ID: 1
Batch ID: 3
Group Member ID: 4
Group member: 1/2

```
--- new messages ---
2017/6/2 16:00:44 - I am the first worker: And I am alone in the batch
```

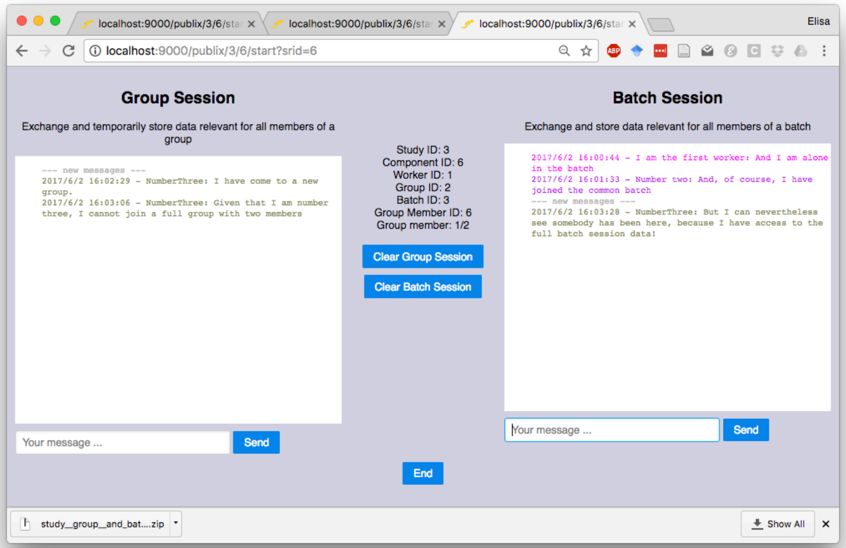
5. Now you have 3 simultaneous study runs. You will notice while writing into the group chat that two of your workers are in the same group - the third one has their own group. Why 2 per group? Because we set the groups to a maximum of 2 members each (page 67). The group chat will use the **Group Session** to allow the 2 members of each group to communicate with each other. Members of other groups will not have access to the chats of this group. However, anything written into the **Batch Session** will be accesssible by all workers that are members of this batch, regardless of the group they're in.


localhost:9000/publix/3/6/start?srid=5
Elisa
Group Session
Exchange and temporarily store data relevant for all members of a group
Batch Session
Exchange and store data relevant for all members of a batch
Your message ...
Send
Clear Group Session
Clear Batch Session
End
study_group_and_bat....zip
Show All

```
2017/6/2 16:00:23 - I am the first worker: I salute you
2017/6/2 16:00:33 - I am the first worker: Nobody to
salute
2017/6/2 16:00:37 - I am the first worker: I am alone in
my group
--- new messages ---
2017/6/2 16:01:14 - Number two: I am number two.
2017/6/2 16:01:21 - Number two: I have joined the first
worker
```

Study ID: 3
Component ID: 6
Worker ID: 1
Group ID: 1
Batch ID: 3
Group Member ID: 5
Group member: 2/2

```
2017/6/2 16:00:44 - I am the first worker: And I am alone
in the batch
--- new messages ---
2017/6/2 16:01:33 - Number two: And, of course, I have
joined the common batch
```


localhost:9000/publix/3/6/start?srid=6
Elisa
Group Session
Exchange and temporarily store data relevant for all members of a group
Batch Session
Exchange and store data relevant for all members of a batch
Your message ...
Send
Clear Group Session
Clear Batch Session
End
study_group_and_bat....zip
Show All

```
--- new messages ---
2017/6/2 16:02:29 - NumberThree: I have come to a new
group.
2017/6/2 16:03:06 - NumberThree: Given that I am number
three, I cannot join a full group with two members
```

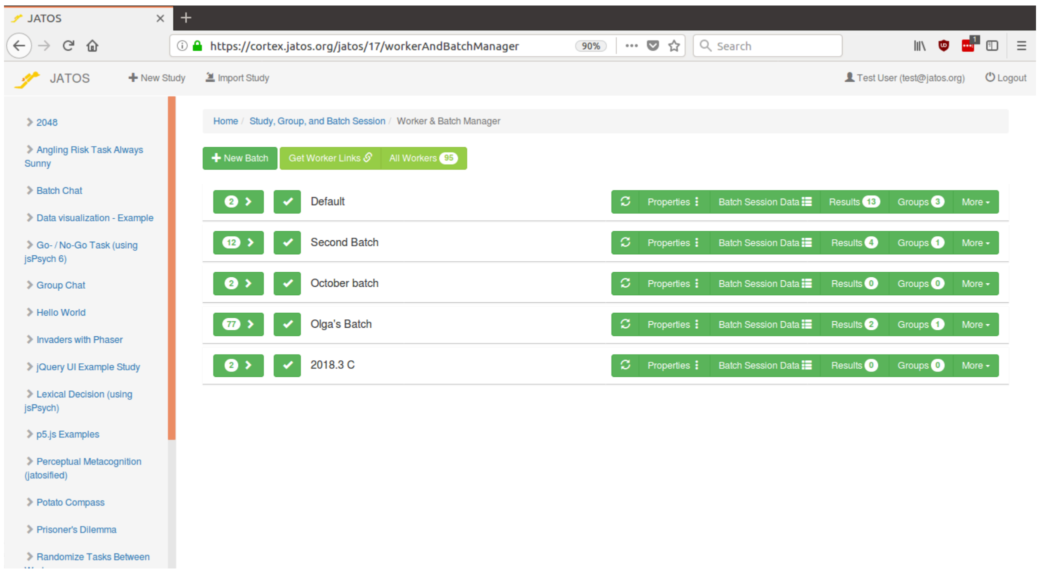
Study ID: 3
Component ID: 6
Worker ID: 1
Group ID: 2
Batch ID: 3
Group Member ID: 6
Group member: 1/2

```
2017/6/2 16:00:44 - I am the first worker: And I am alone
in the batch
2017/6/2 16:01:33 - Number two: And, of course, I have
joined the common batch
--- new messages ---
2017/6/2 16:03:28 - NumberThree: But I can nevertheless
see somebody has been here, because I have access to the
full batch session data!
```

Run your Study with Worker & Batch Manager

Worker & Batch Manager

The Worker & Batch Manager is the place where you generate links for your particpants to run the your study, organize them into Batches and handle their results.



This is a screenshot of JATOS v3.3.1. In earlier versions it was called Batch Manager and looked a bit simpler. Each row represents a batch which in turn is a collection of workers.

How to let participants run your study: Workers

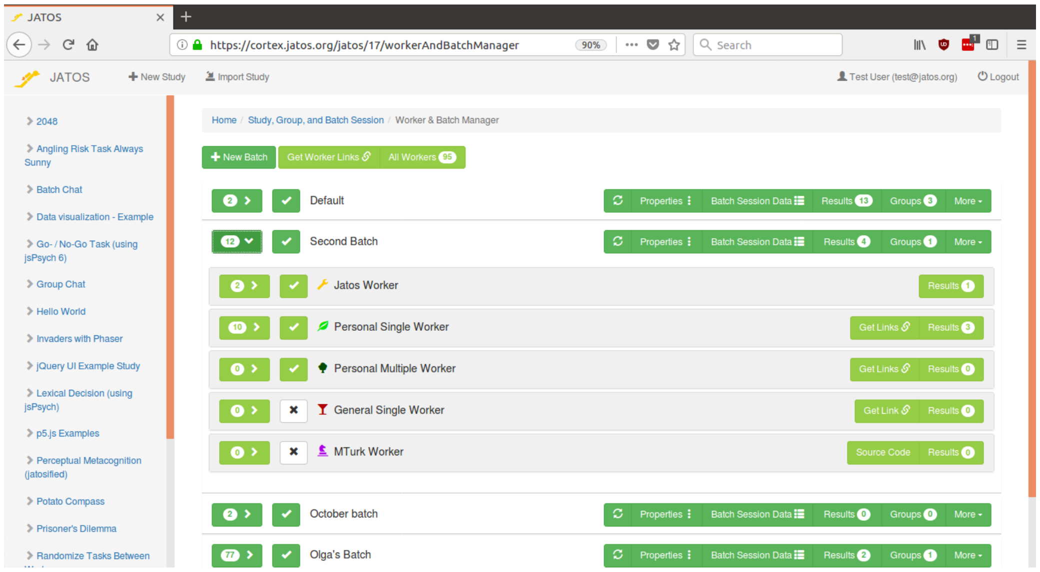
During development of your study you would usually run it with the “Run” button in the study page. But then, when you are done developing you want to let others run your study - you want participants (or workers as they are called in JATOS) do it. For this JATOS lets you create links that you can hand out to your workers (e.g. via email or social media).

Generate links and hand them to your workers

JATOS has different worker types (each with different properties). That’s well explained in a dedicated page: Worker Types (page 38).

Click on the “❯” button in the left in each batch row to expand the Worker Setup.

Worker Setup



Screenshot of a Worker & Batch Mangager with an open Worker Setup for the second batch. In JATOS version < 3.3.1 it is reachable via the "Worker Setup" button.

The Worker Setup is the place where you generate or view (for Jatos and MTurk workers) the links for all workers types.

Get Links

For **Personal Single Workers** and **Personal Multiple Workers** click "**Get Links** 🔗" ("Add" in older versions). You can enter a description or identification for the worker in the 'Comments' box. You can also create several at once.

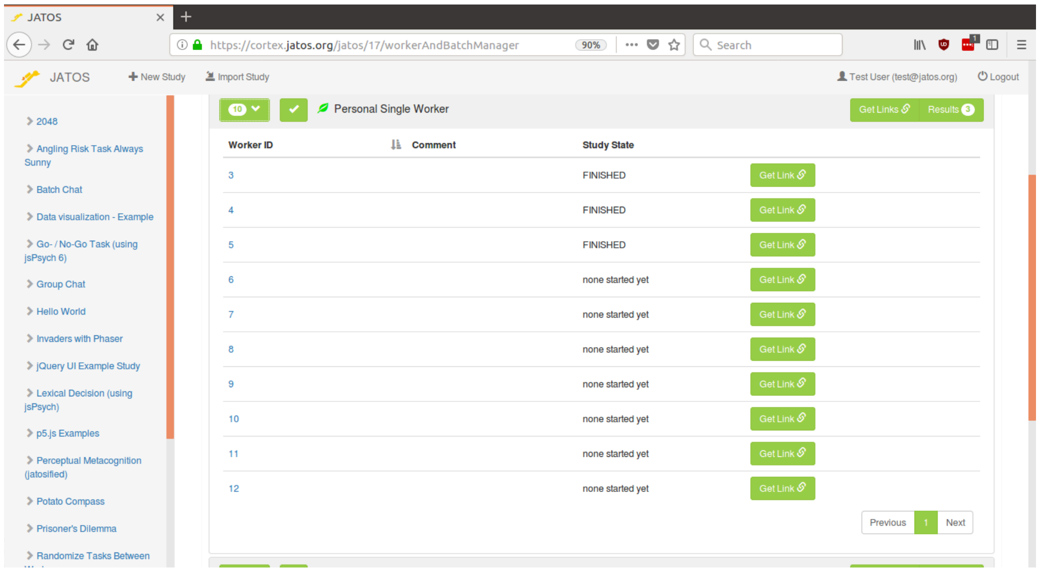
General Single Workers only have one link. Each time somebody clicks on the link, JATOS will create a new separate worker. Get this link by clicking on "**Get Link** 🔗" in its row.

How to connect to MTurk and create links to run with **MTurk Workers** is described in its own page: Connect to Mechanical Turk (page 43).

Alternatively there is a "**Get Worker Links** 🔗" button in the top of the Worker & Batch Manger page that is a shortcut to create those links.

See Workers

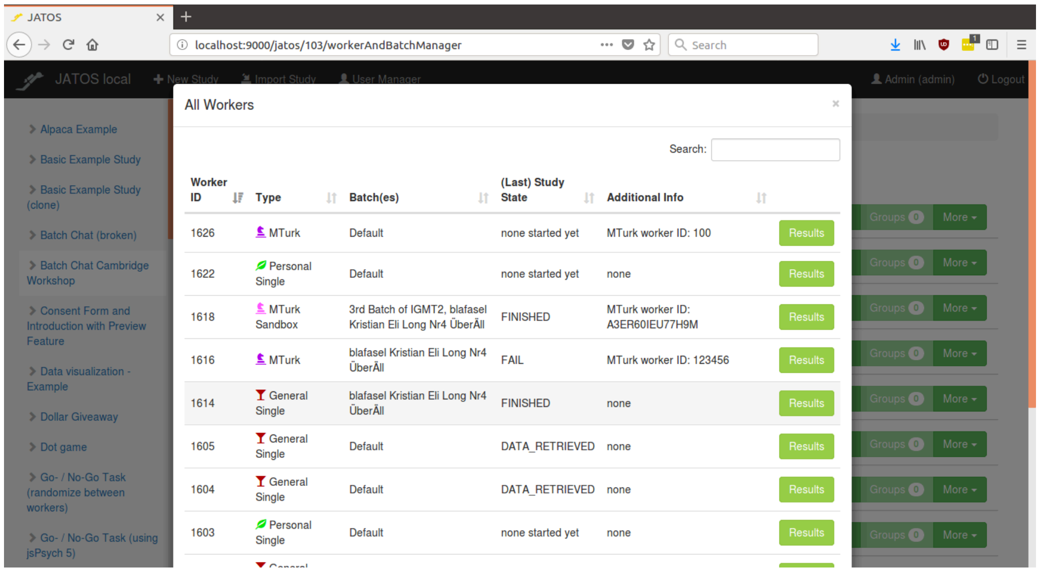
Click on the “ ❯” button in the left in each worker type row to expand it and see all generated workers. The column “Study State” indicates in which state (page 48) this study run currently is.


JATOS
https://cortex.jatos.org/jatos/17/workerAndBatchManager
JATOS
New Study
Import Study
Test User (test@jatos.org)
Logout
2048
Angling Risk Task Always Sunny
Batch Chat
Data visualization - Example
Go- / No-Go Task (using jsPsych 6)
Group Chat
Hello World
Invaders with Phaser
jQuery UI Example Study
Lexical Decision (using jsPsych)
p5.js Examples
Perceptual Metacognition (jatosified)
Potato Compass
Prisoner's Dilemma
Randomize Tasks Between
Personal Single Worker
Get Links
Results 3
Previous
1
Next

Worker ID	Comment	Study State	
3		FINISHED	Get Link
4		FINISHED	Get Link
5		FINISHED	Get Link
6		none started yet	Get Link
7		none started yet	Get Link
8		none started yet	Get Link
9		none started yet	Get Link
10		none started yet	Get Link
11		none started yet	Get Link
12		none started yet	Get Link

Screenshot with expanded Personal Single Worker

Another way to see your workers is the button “**All Workers**” in the top of the Worker & Batch Manager page.


JATOS
localhost:9000/jatos/103/workerAndBatchManager
JATOS local
New Study
Import Study
User Manager
Admin (admin)
Logout
Alpaca Example
Basic Example Study
Basic Example Study (clone)
Batch Chat (broken)
Batch Chat Cambridge Workshop
Consent Form and Introduction with Preview Feature
Data visualization - Example
Dollar Giveaway
Dot game
Go- / No-Go Task (randomize between workers)
Go- / No-Go Task (using jsPsych 5)
All Workers
Search:

Worker ID	Type	Batch(es)	(Last) Study State	Additional Info	
1626	MTurk	Default	none started yet	MTurk worker ID: 100	Results
1622	Personal Single	Default	none started yet	none	Results
1618	MTurk Sandbox	3rd Batch of IGMT2, blafasel Kristian Eli Long Nr4 ÜberÄll	FINISHED	MTurk worker ID: A3ER60IEU77H9M	Results
1616	MTurk	blafasel Kristian Eli Long Nr4 ÜberÄll	FAIL	MTurk worker ID: 123456	Results
1614	General Single	blafasel Kristian Eli Long Nr4 ÜberÄll	FINISHED	none	Results
1605	General Single	Default	DATA_RETRIEVED	none	Results
1604	General Single	Default	DATA_RETRIEVED	none	Results
1603	Personal Single	Default	none started yet	none	Results

Screenshot of All Workers table: Here one can search and filter through all workers of all batches and all types that belong to this study.

How to organize your workers: Batches

A batch is a collection of workers together with some properties. Using different batches is useful to organize your study runs, separate their results and vary their setup. E.g. you could separate a pilot run from the “proper” experiment, or you could use different batches for different worker types.

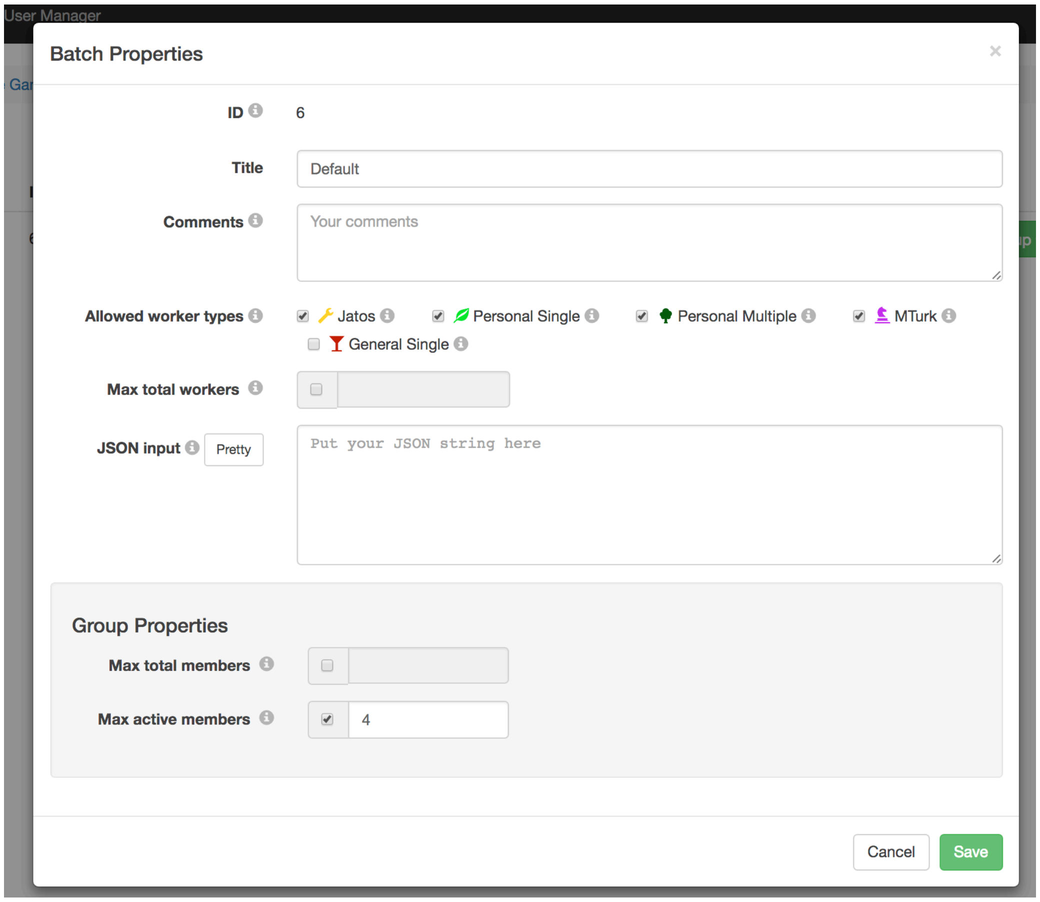
Batches are organized in the Worker & Batch Manager. Here you can create and delete batches, access each batch’s properties and edit its **Batch Session Data** or look through their results.

Each study comes with a “Default” batch.

You can **deactivate** or **activate** a batch by clicking on the checkbox button in each batch row. A deactivated batch doesn’t allow any study runs.

Batch Properties

For each batch, you can limit the maximum number of workers that will ever be able to run a study in this batch by setting the **Maximum Total Workers**.


Batch Properties

ID	6
Title	Default
Comments	Your comments
Allowed worker types	✓ Jatos ✓ Personal Single ✓ Personal Multiple ✓ MTurk ☐ General Single
Max total workers	☐
JSON input (Pretty)	Put your JSON string here

Group Properties

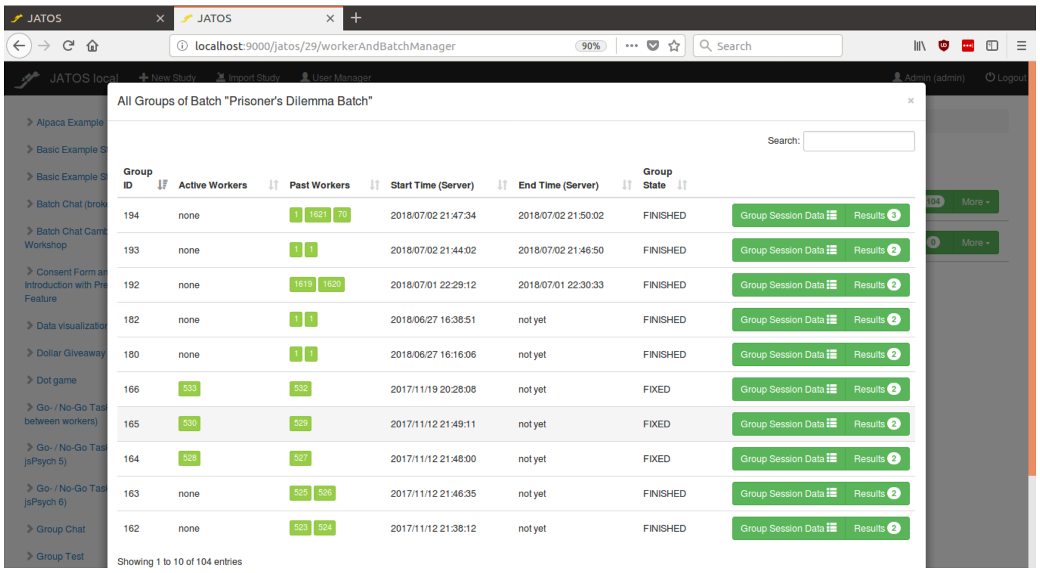
Max total members	☐
Max active members	✓ 4

Additionally you can switch on or off worker types in the **Allowed Worker Types**. Unchecked worker types are not allowed to run a study. Always check before you send out links to study runs that the corresponding worker types are switched on.

The **Group Properties** relate to group studies (page 67).

Groups (since v3.3.1)

A batch is also the place where JATOS groups (page 67) are handled. Here you can an get an overview of the Groups that belong to this batch: see what their member workers are or edit the **Group Session Data**.



All Groups of Batch "Prisoner's Dilemma Batch"

Search:

Group ID	Active Workers	Past Workers	Start Time (Server)	End Time (Server)	Group State	
194	none	1 1621 70	2018/07/02 21:47:34	2018/07/02 21:50:02	FINISHED	Group Session Data Results 3
193	none	1 1	2018/07/02 21:44:02	2018/07/02 21:46:50	FINISHED	Group Session Data Results 2
192	none	1619 1620	2018/07/01 22:29:12	2018/07/01 22:30:33	FINISHED	Group Session Data Results 2
182	none	1 1	2018/06/27 16:38:51	not yet	FINISHED	Group Session Data Results 2
180	none	1 1	2018/06/27 16:16:06	not yet	FINISHED	Group Session Data Results 2
166	533	532	2017/11/19 20:28:08	not yet	FIXED	Group Session Data Results 2
165	530	529	2017/11/12 21:49:11	not yet	FIXED	Group Session Data Results 2
164	528	527	2017/11/12 21:48:00	not yet	FIXED	Group Session Data Results 2
163	none	525 526	2017/11/12 21:46:35	not yet	FINISHED	Group Session Data Results 2
162	none	523 524	2017/11/12 21:38:12	not yet	FINISHED	Group Session Data Results 2

Showing 1 to 10 of 104 entries

Screenshot of an Groups table (available in JATOS version >= 3.3.1): "Active Workers" are the workers that are currently members in the group, "Past Workers" the ones that where member at one point in the past. "Results" shows only the study results that belong to this group. "Group State" can be START, FINISHED, or FIXED.

Worker Types

Summary: Details of different worker types

Overview

Following Amazon Mechanical Turk's terminology, a worker in JATOS is a person who runs a study. Different worker types access a study in different ways. For example, some workers can run the same study multiple times, whereas others can do it only once.

	Jatos	Personal Single	Personal Multiple	General Single	General Multiple (since v3.3.2)	MTurk
Icon	🔧	🍃	🌳	🍸	✱	♞
Typical use	During study development	Small targeted group, each one of them gets a link	Small targeted group of workers who pilot the study or need to do it multiple times	Bigger groups but with less control; link shared e.g. via social media	Bigger groups and where the workers need to do it multiple times	For Amazon Mechanical Turk
Created when?	Together with the JATOS user	By yourself in the Worker Setup	By yourself in the Worker Setup	On-the-fly whenever someone uses the link	On-the-fly whenever someone uses the link	On-the-fly after a MTurk worker clicked on the HIT link
Repeat the same study with the same link	(has no links)	✗	✓ (keeps the same worker)	✗	✓ (creates a new worker each time)	✗

	Jatos	Personal Single	Personal Multiple	General Single	General Multiple (since v3.3.2)	MTurk
Run different studies with the same worker	✓	✗	✗	✗	✗	✓
Supports preview of studies (page 41)	✗	✓	✗	✓	✗	✗
Possible bulk creation	✗	✓	✓	✗	✗	✗
Run group studies (page 0)	✓	✓	✓	✓	✓	✓

Jatos Worker

Jatos workers can run any study as many times as they want.

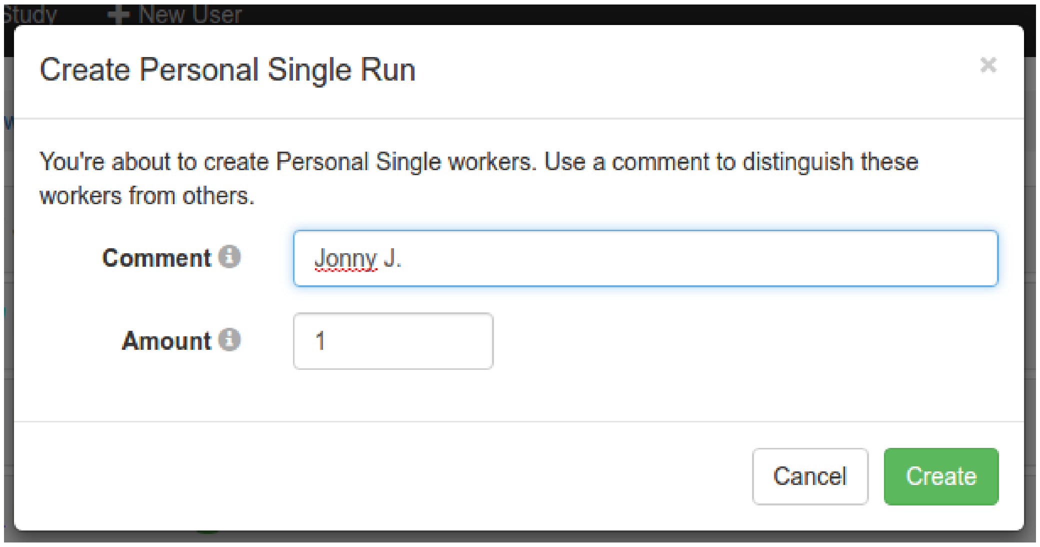
Jatos workers run a study (or any of its components individually) by clicking on the *Run* buttons in the GUI. Jatos workers are usually the **researchers trying out their own studies**. Each JATOS user (i.e., anybody with a JATOS login) has their own Jatos worker.

Personal Single Worker

With a Personal Single link **a study can be run only once** (*But see Preview Links (page 41)). You can think of them as *personalized links with single access*. Each Personal Single link corresponds a Personal Single worker.

Usually you would send a Personal Single link to workers that you contact individually. Each link can be personalized with a **Comment** you provide while creating it (e.g. by providing a pseudonym).

Personal Single links are useful in small studies, where it's feasible to contact each worker individually, or (e.g.) you want to be able to pair up several results (either from the same or different studies) in a longitudinal design. You can create Personal Single links in bulk by changing the **Amount** value.

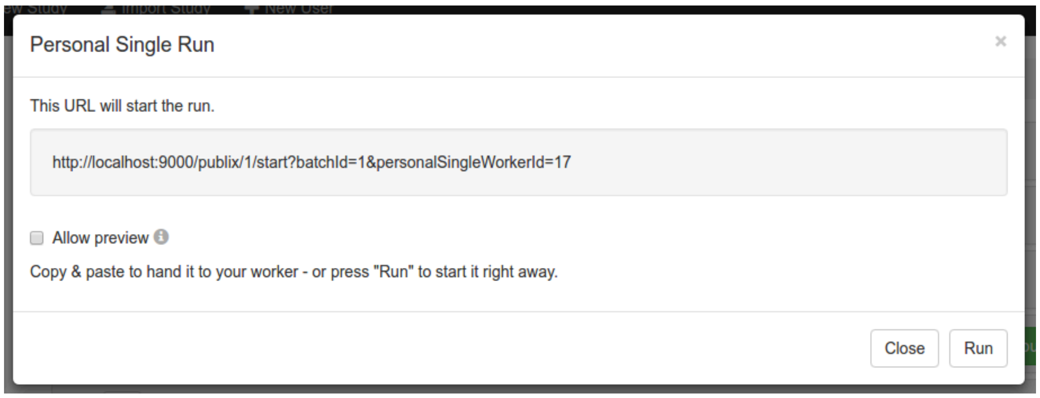


Create Personal Single Run

You're about to create Personal Single workers. Use a comment to distinguish these workers from others.

Comment: Jonny J.

Amount: 1



Personal Single Run

This URL will start the run.

```
http://localhost:9000/publix/1/start?batchId=1&personalSingleWorkerId=17
```

Allow preview

Copy & paste to hand it to your worker - or press "Run" to start it right away.

Personal Multiple Worker

With a Personal Multiple link the worker can **run a study as many times as they want**. Each Personal Multiple link corresponds to a Personal Multiple worker.

You could send *Personal Multiple* links to your pilot workers. Each link can be personalized with a **Comment** you provide while creating it (e.g. by providing a pseudonym). You can create Personal Multiple links in bulk by changing the **Amount** value.

General Single Worker

This link type can be used **many times by different participants to run a study but only once per browser** (*But see Preview Links (page 41)). Each time the link is used a new General Single worker is created on-the-fly.

You could distribute a *General Single* link through twitter, a mailing list or posting it on a public website. It is essentially useful for cases where you want to collect data from a large number of workers.

Keep in mind, however, that JATOS uses the browser's cookies to decide whether a worker has already accessed a study. If someone uses a different computer, a new browser, or simply deletes their browser's cookies, then JATOS will assume that it's a new worker. So the same person could (with some effort) use a General Single link several times.

General Multiple Worker (since version 3.3.2)

A General Multiple link is the least restrictive type and can be used **many times by different participants to run a study**. The difference to a General Single is that the General Multiple link can be used repeatedly **even in the same browser**. Each time a General Multiple link is used a new General Multiple worker is created on-the-fly.

MTurk Worker

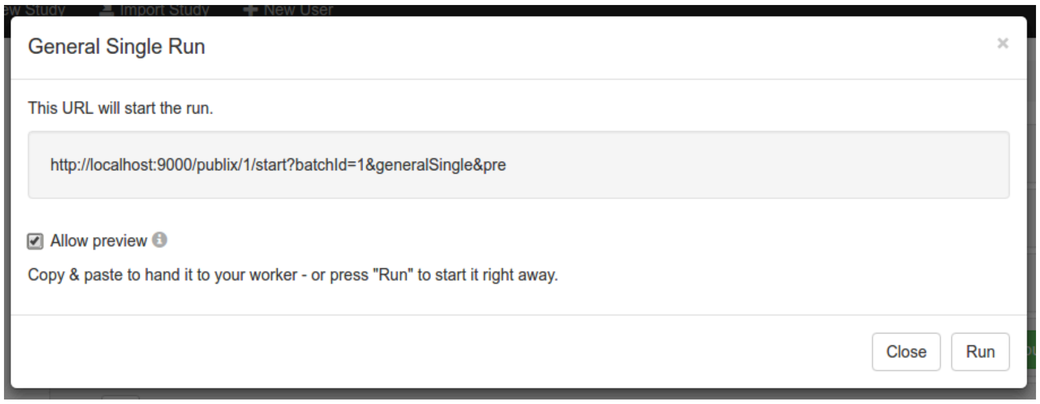
MTurk workers access a study **only once**, through a link in Amazon's Mechanical Turk (AMT). Each MTurk worker in JATOS corresponds to a single worker in AMT.

DATA PRIVACY NOTE: If the same worker from AMT does two of your studies, the two results will be paired with the same MTurk worker in JATOS. This means that you could gather data from different studies, without your workers ever consenting to it. For this reason, we recommend that you delete your data from JATOS as soon as you finish a study. This way, if the same worker from AMT takes part in a different study, they will get a new MTurk worker, and you will not be able to automatically link their data between different studies. See our Data Privacy and Ethics (page 0) page for more details on this.

Preview Links

A normal **General Single** or **Personal Single** link is restrictive: once a worker clicked on the link - that's it. JATOS will not let them run the study twice. But in some cases you may want to distribute a link and let your workers preview the first component of your study (where you could e.g. describe what they will have to do, how long it will take, ask for consent, etc). Often, workers see this description and decide that they want to do the study later.

To allow them to do this, activate the checkbox **Allow preview** (this will add a `&pre` to the end of the URL). Now your workers can use the link as often as they want - as long as they don't go further than the first component. But once the second component is started, JATOS will restrict access to the study in the usual way as it is for General Single and Personal Single workers. This means that you'll get an error if you try to use the link again to access the study.



General Single Run

This URL will start the run.

http://localhost:9000/publix/1/start?batchId=1&generalSingle&pre

☑ Allow preview

Copy & paste to hand it to your worker - or press "Run" to start it right away.

Connect to Mechanical Turk

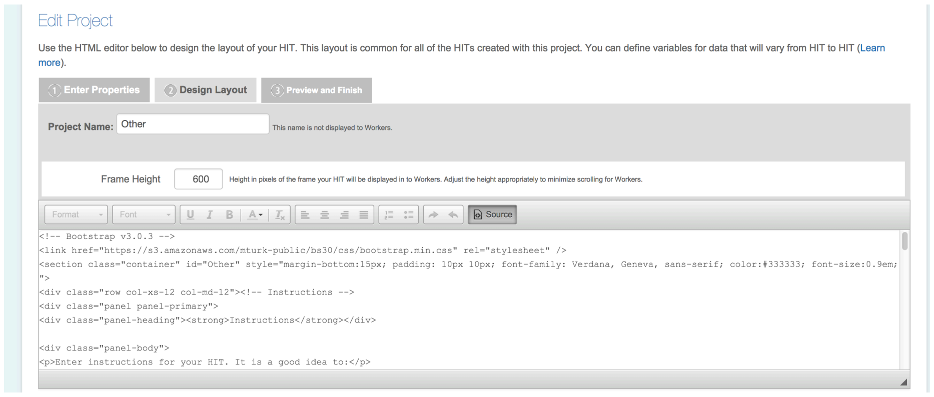
Connecting your JATOS study to the Mturk is easy, although a fair amount of clicking is required.

You will need:

- A requester Mturk account
- Your study running on JATOS
- A description of the study (this can be the same as the one you included in the study description within JATOS)

The steps to create a project are part of the MTurk interface.

1. Create –> New Project –> Survey Link –> Create Project
2. Complete the 'Enter Properties' tab
3. Click on the 'Design layout' button in the bottom of the page.
4. Click on the 'Source' button. You'll see some text in an editable window, corresponding to an html file. Delete the entire text in this field.


Edit Project

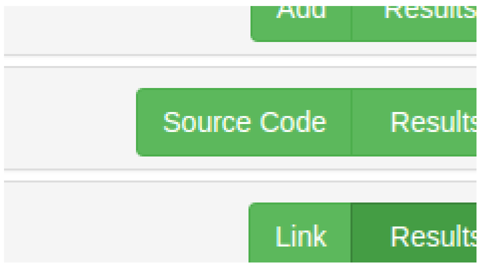
Use the HTML editor below to design the layout of your HIT. This layout is common for all of the HITs created with this project. You can define variables for data that will vary from HIT to HIT (Learn more).

Project Name: Other This name is not displayed to Workers.

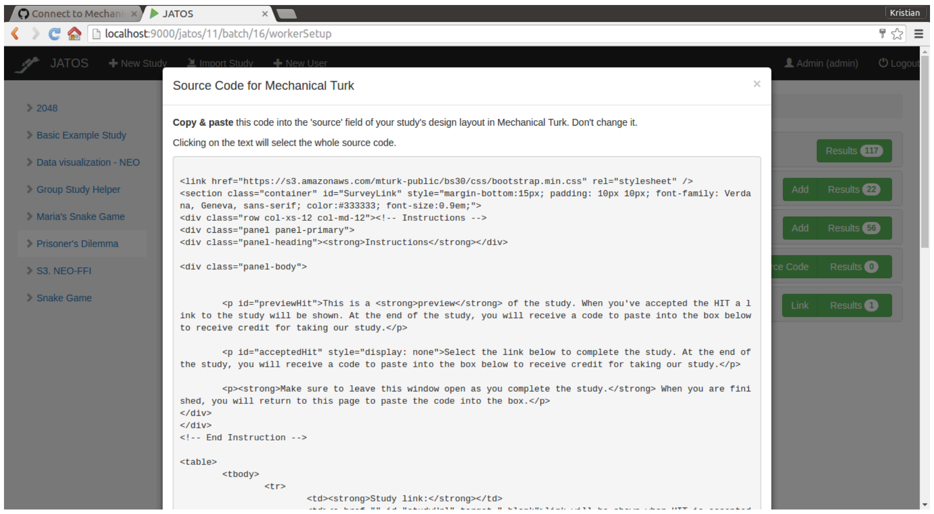
Frame Height 600 Height in pixels of the frame your HIT will be displayed in to Workers. Adjust the height appropriately to minimize scrolling for Workers.

```
<!-- Bootstrap v3.0.3 -->
<link href="https://s3.amazonaws.com/mturk-public/bs30/css/bootstrap.min.css" rel="stylesheet" />
<section class="container" id="Other" style="margin-bottom:15px; padding: 10px 10px; font-family: Verdana, Geneva, sans-serif; color:#333333; font-size:0.9em;
">
<div class="row col-xs-12 col-md-12"><!-- Instructions -->
<div class="panel panel-primary">
<div class="panel-heading"><strong>Instructions</strong></div>

<div class="panel-body">
<p>Enter instructions for your HIT. It is a good idea to:</p>
```

5. In JATOS, go to the Study Toolbar —> Worker & Batch Manager —> Worker Setup —> MTurk Worker row —> Source Code.



6. You'll see a box with HTML code, similar to the one shown here. Copy paste the code from JATOS to the MTurk source section.


Source Code for Mechanical Turk

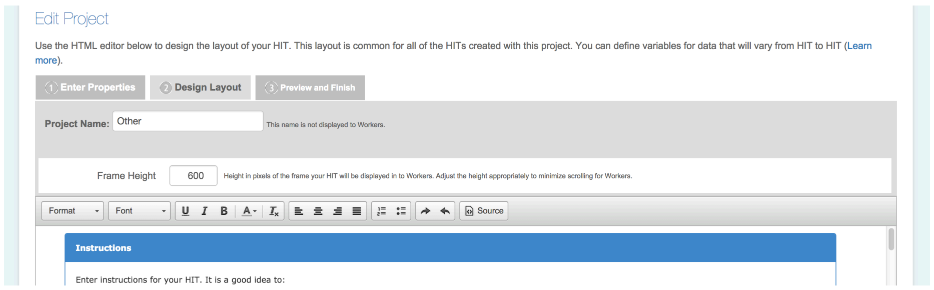
Copy & paste this code into the 'source' field of your study's design layout in Mechanical Turk. Don't change it.

Clicking on the text will select the whole source code.

```
<link href="https://s3.amazonaws.com/mturk-public/bs30/css/bootstrap.min.css" rel="stylesheet" />
<section class="container" id="SurveyLink" style="margin-bottom:15px; padding: 10px 10px; font-family: Verda
na, Geneva, sans-serif; color:#333333; font-size:0.9em;">
<div class="row col-xs-12 col-md-12"><!-- Instructions -->
<div class="panel panel-primary">
<div class="panel-heading"><strong>Instructions</strong></div>

<div class="panel-body">


        <p id="previewHit">This is a <strong>preview</strong> of the study. When you've accepted the HIT a l
ink to the study will be shown. At the end of the study, you will receive a code to paste into the box below
to receive credit for taking our study.</p>

        <p id="acceptedHit" style="display: none">Select the link below to complete the study. At the end of
the study, you will receive a code to paste into the box below to receive credit for taking our study.</p>

        <p><strong>Make sure to leave this window open as you complete the study.</strong> When you are fini
shed, you will return to this page to paste the code into the box.</p>
</div>
</div>
<!-- End Instruction -->

<table>
        <tbody>
                <tr>
                        <td><strong>Study link:</strong></td>
```

7. This HTML code works out-of-the-box and you don't have to change anything (but you can if you want).

8. Back in MTurk click on the 'Source' button again, and continue setting up your study in MTurk.


Edit Project

Use the HTML editor below to design the layout of your HIT. This layout is common for all of the HITs created with this project. You can define variables for data that will vary from HIT to HIT (Learn more).

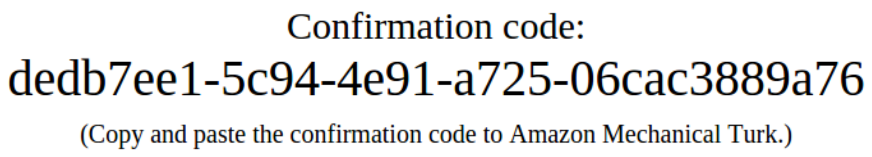
Project Name: Other — This name is not displayed to Workers.

Frame Height: 600 — Height in pixels of the frame your HIT will be displayed in to Workers. Adjust the height appropriately to minimize scrolling for Workers.

Instructions

Enter instructions for your HIT. It is a good idea to:

9. When an MTurk worker finishes a study they'll see a confirmation code. To assign payment to individual workers, just compare the confirmation codes stored in JATOS' results view to those stored in MTurk.


Confirmation code:

dedb7ee1-5c94-4e91-a725-06cac3889a76

(Copy and paste the confirmation code to Amazon Mechanical Turk.)

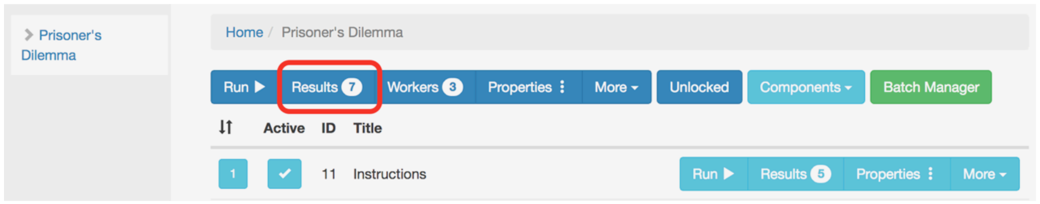
See the Tips & Tricks (page 52) page for some useful information on how to test your study before officially posting it on MTurk.

Manage Results

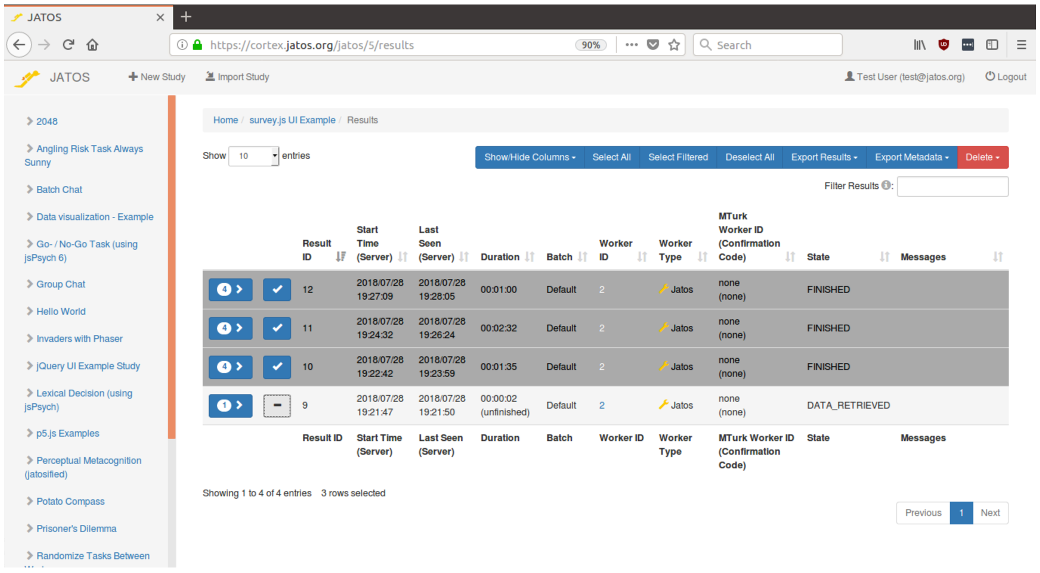
Summary: The results pages (study, component, batch or worker) present all the data that were collected during the study runs, including the result data and metadata (e.g. worker ID, start time etc.).

Results Pages

Once you collected data for a study, you can see and manage the results by clicking on one of the *Results* buttons.



The image below is an example of a study results page. There's quite a lot of information here, so we'll go through each piece.



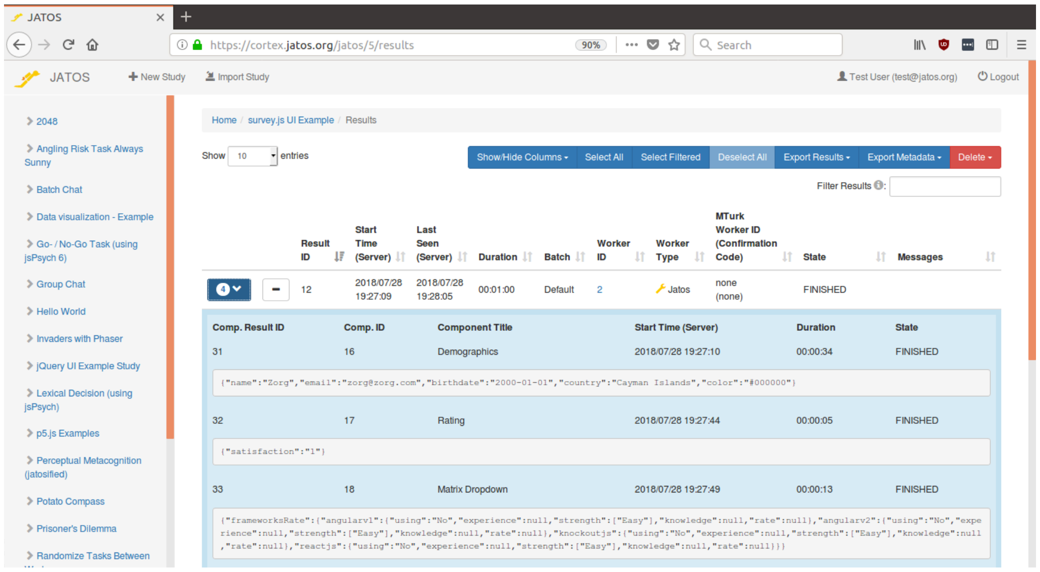
		Result ID	Start Time (Server)	Last Seen (Server)	Duration	Batch	Worker ID	Worker Type	MTurk Worker ID (Confirmation Code)	State	Messages
4	✓	12	2018/07/28 19:27:09	2018/07/28 19:28:05	00:01:00	Default	2	Jatos	none (none)	FINISHED	
4	✓	11	2018/07/28 19:24:32	2018/07/28 19:26:24	00:02:32	Default	2	Jatos	none (none)	FINISHED	
4	✓	10	2018/07/28 19:22:42	2018/07/28 19:23:59	00:01:35	Default	2	Jatos	none (none)	FINISHED	
1	−	9	2018/07/28 19:21:47	2018/07/28 19:21:50	00:00:02 (unfinished)	Default	2	Jatos	none (none)	DATA_RETRIEVED	
		Result ID	Start Time (Server)	Last Seen (Server)	Duration	Batch	Worker ID	Worker Type	MTurk Worker ID (Confirmation Code)	State	Messages

Showing 1 to 4 of 4 entries 3 rows selected

Interacting With The Results Table

View Result Data

Each study result has an arrow on the left. If you click on it, the result data for this study run will be displayed underneath the row. Since a study can have several components and each component produces its own result data there can be several result data each in its own row (like in the screenshot below).



		Result ID	Start Time (Server)	Last Seen (Server)	Duration	Batch	Worker ID	Worker Type	MTurk Worker ID (Confirmation Code)	State	Messages
4		12	2018/07/28 19:27:09	2018/07/28 19:28:05	00:01:00	Default	2	Jatos	none (none)	FINISHED	

Comp. Result ID	Comp. ID	Component Title	Start Time (Server)	Duration	State
31	16	Demographics	2018/07/28 19:27:10	00:00:34	FINISHED

```
{"name":"Zorg","email":"zorg@zorg.com","birthdate":"2000-01-01","country":"Cayman Islands","color":"#000000"}
```

Comp. Result ID	Comp. ID	Component Title	Start Time (Server)	Duration	State
32	17	Rating	2018/07/28 19:27:44	00:00:05	FINISHED

```
{"satisfaction":"1"}
```

Comp. Result ID	Comp. ID	Component Title	Start Time (Server)	Duration	State
33	18	Matrix Dropdown	2018/07/28 19:27:49	00:00:13	FINISHED

```
{"frameworksRate":{"angularv1":{"using":"No","experience":null,"strength":["Easy"],"knowledge":null,"rate":null},"angularv2":{"using":"No","expe
rience":null,"strength":["Easy"],"knowledge":null,"rate":null},"knockoutjs":{"using":"No","experience":null,"strength":["Easy"],"knowledge":null
,"rate":null},"reactjs":{"using":"No","experience":null,"strength":["Easy"],"knowledge":null,"rate":null}}}
```

Selecting Results

Prior to JATOS version 3.3.1 you could select/deselect a specific result by clicking anywhere on the row. Selected results change color to dark blue.

From 3.3.1 on there is a checkbox on the left side of each row.

You can also use the buttons on the bar above to select/deselect all results in the table or select only all filtered ones.

Filter Results

The filter lets you search all all fields in the results table (the metadata) and from JATOS version 3.3.1 on also in the result data themselves.

If you type, for example, “Personal Single” in the *Filter Results* field, only the results ran by a Personal Single worker will appear on the table. You can then click on *Select Filtered* to select and then export only those results that you’re interested in.

Export Result Data

Once you selected the results you're interested in, click *Export Results* and your browser will download a text file that contains your results in whatever format (text, csv, json) you stored them. Then read this text file with SPSS, Excel, Matlab, R or whatever program you use to analyze results.

Export Metadata (JATOS version >= 3.3.1)

Sometimes one is also interested in the metadata, that is what's in the actual table fields ("Result ID", "Start Time" , "Last Seen", …). For this click on *Export Metadata* and the metadata of the selected results will be downloaded in CSV format.

Delete Results

You can click *Delete* to remove all or only some selected results (result data + metadata). Keep in mind **there's no undo function for this**.

Table Columns

You can show and hide the columns displayed in the table with the drop-down menu under the *Display Columns* button.

Result ID

An identifier assigned by JATOS to each study result. A study result is actually a set of component results, each of them with their own (different) *Component Result ID*.

Start Time

Time (set at the server's time zone) at which the first component of the study was started.

End Time

Time (set at the server's time zone) at which the last component of the study was finished.

Last Seen

Each component running in a worker's browser sends a "heartbeat" regularly back to JATOS. Last Seen is the time of the last heartbeat received. The heartbeat stops either when the study is finished or when the browser tab is closed. The default period of the heartbeat is 2 minutes but you can change it through a jatos.js function (page 106).

Duration

Simply the time difference between the start and end time.

Batch

Name of the batch the worker belongs to.

Worker ID

Assigned by JATOS. Each worker has its own Worker ID. JATOS' admin user will always have Worker ID 1. You can click on a Worker ID to see all the worker's results.

Worker Type

Displays the Worker type (page 38) that ran the study.

MTurk Worker ID (Confirmation Code)

Only applies to studies run by MTurk workers. An identifier given by Amazon Mechanical Turk's, not by JATOS. The Confirmation Code is generated by JATOS and given to the worker as proof of his work.

Group ID

Only available for group studies. It identifies the group.

State

Possible states for study results are:

- PRE - Preview of study (page 41) (exists only in PersonalSingleWorker and GeneralSingleWorker)
- STARTED - Study was started

- DATA_RETRIEVED - Study's jsonData were retrieved
- FINISHED - Study successfully finished
- ABORTED - Study aborted by worker
- FAIL - Something went wrong

Possible states for component results are:

- STARTED - Component was started
- DATA_RETRIEVED - Component's jsonData were retrieved
- RESULTDATA_POSTED - Result data were posted
- FINISHED - Component was finished
- RELOADED - Component was reloaded
- ABORTED - Component aborted by worker
- FAIL - Something went wrong

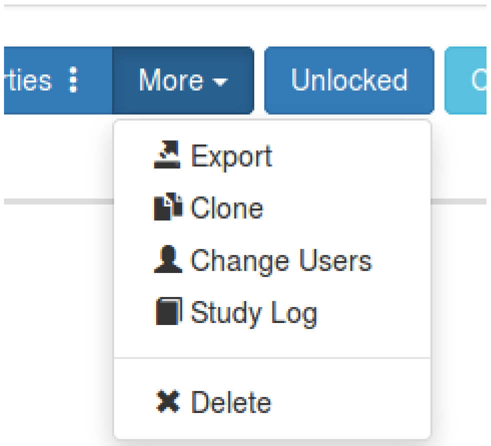
Messages

A message that can be set together with jatos.endStudy (page 111) or jatos.abortStudy (page 110).

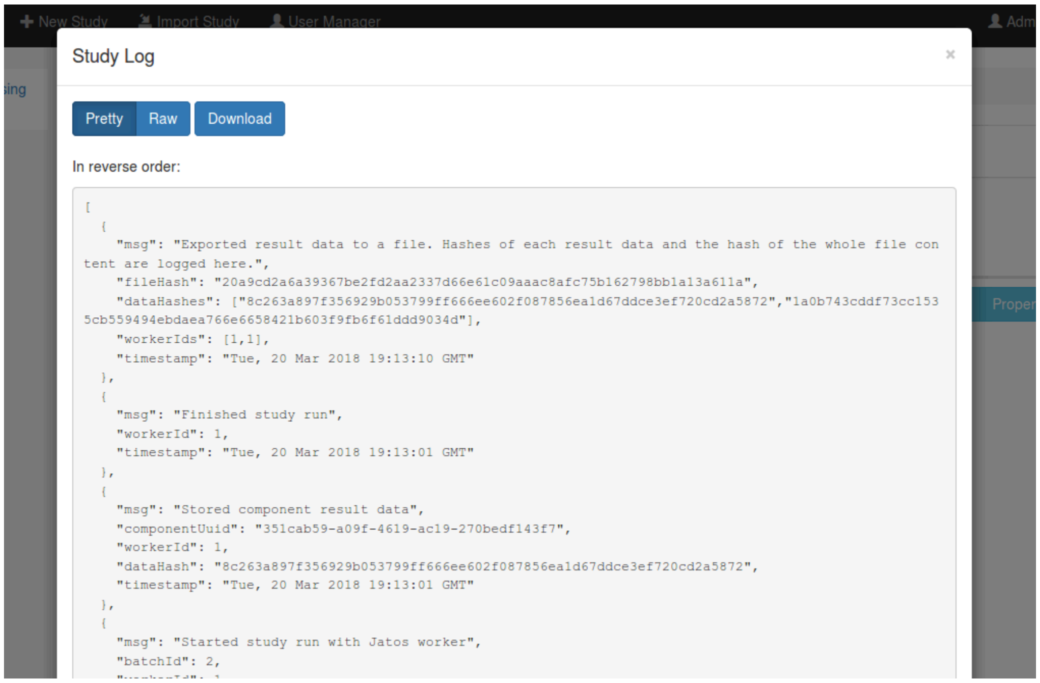
Study Log

From version 3.2.1 onwards JATOS stores a log file for each study (not to be confused with JATOS' log). This file has a line for every relevant event that happened in a study, most importantly when a component result was saved, exported or deleted. Also, it contains a hash – a string that is generated by the contents of the result data itself. This, in principle, would allow any JATOS user to show that the data have not been modified, and that no result was deleted between data collection and publication.

You can see the log by clicking on **More** in the study toolbar and then **Study Log**:



Then the log looks similar to this:

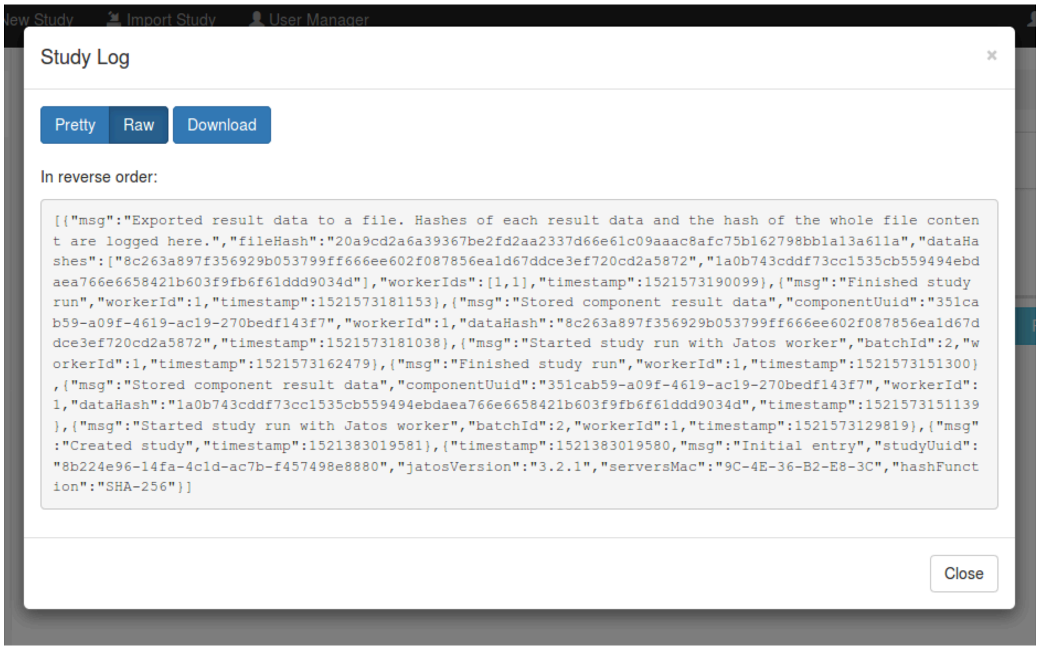


In reverse order:

```
[
  {
    "msg": "Exported result data to a file. Hashes of each result data and the hash of the whole file con
tent are logged here.",
    "fileHash": "20a9cd2a6a39367be2fd2aa2337d66e61c09aaac8afc75b162798bb1a13a611a",
    "dataHashes": ["8c263a897f356929b053799ff666ee602f087856ea1d67ddce3ef720cd2a5872","1a0b743cddf73cc153
5cb559494ebdaea766e6658421b603f9fb6f61ddd9034d"],
    "workerIds": [1,1],
    "timestamp": "Tue, 20 Mar 2018 19:13:10 GMT"
  },
  {
    "msg": "Finished study run",
    "workerId": 1,
    "timestamp": "Tue, 20 Mar 2018 19:13:01 GMT"
  },
  {
    "msg": "Stored component result data",
    "componentUuid": "351cab59-a09f-4619-ac19-270bedf143f7",
    "workerId": 1,
    "dataHash": "8c263a897f356929b053799ff666ee602f087856ea1d67ddce3ef720cd2a5872",
    "timestamp": "Tue, 20 Mar 2018 19:13:01 GMT"
  },
  {
    "msg": "Started study run with Jatos worker",
    "batchId": 2,
```

A few more details:

- The study log won't be necessary in most cases. Just nice to have. Just in case.
- In the GUI you will see only the **last 100 entries** of the study log but you can get the whole log by downloading it. In the GUI the log is in **reversed** order - the downloaded one has normal order.
- The following events are logged: **create/delete study**, **run/finish study**, **save/delete/export results**.
- In case of result exports, additionally to hashes for single component result data, the hash of the whole exported file is being logged. Since a hash changes if a result is altered or deleted, this can prove **data integrity** should it ever being questioned. It uses the hash algorithm SHA-256.
- The study log is only as safe as the server machine on which JATOS is running. Anybody with access to the server can potentially modify the study log file and e.g. hide that data has been deleted. We can't prevent this, so it's important to have a safe server that only admins can access.
- The study log is in JSON format. Choose between **pretty** (like in the screenshot above) or **raw** (in the one below).



In reverse order:

```
[{"msg":"Exported result data to a file. Hashes of each result data and the hash of the whole file conten
t are logged here.","fileHash":"20a9cd2a6a39367be2fd2aa2337d66e61c09aaac8afc75b162798bb1a13a611a","dataHa
shes":["8c263a897f356929b053799ff666ee602f087856ea1d67ddce3ef720cd2a5872","1a0b743cddf73cc1535cb559494ebd
aea766e6658421b603f9fb6f61ddd9034d"],"workerIds":[1,1],"timestamp":1521573190099},{"msg":"Finished study
run","workerId":1,"timestamp":1521573181153},{"msg":"Stored component result data","componentUuid":"351ca
b59-a09f-4619-ac19-270bedf143f7","workerId":1,"dataHash":"8c263a897f356929b053799ff666ee602f087856ea1d67d
dce3ef720cd2a5872","timestamp":1521573181038},{"msg":"Started study run with Jatos worker","batchId":2,"w
orkerId":1,"timestamp":1521573162479},{"msg":"Finished study run","workerId":1,"timestamp":1521573151300}
,{"msg":"Stored component result data","componentUuid":"351cab59-a09f-4619-ac19-270bedf143f7","workerId":
1,"dataHash":"1a0b743cddf73cc1535cb559494ebdaea766e6658421b603f9fb6f61ddd9034d","timestamp":1521573151139
},{"msg":"Started study run with Jatos worker","batchId":2,"workerId":1,"timestamp":1521573129819},{"msg"
:"Created study","timestamp":1521383019581},{"timestamp":1521383019580,"msg":"Initial entry","studyUuid":
"8b224e96-14fa-4c1d-ac7b-f457498e8880","jatosVersion":"3.2.1","serversMac":"9C-4E-36-B2-E8-3C","hashFunct
ion":"SHA-256"}]
```

Tips & Tricks

Run up to 10 studies in the same browser at the same time

When a participant runs a study they usually run only one at any given time. For them it's not necessary to run more than one study in parallel in the same browser. But during development of a study it can be an immensely useful feature especially if you are using the Batch Session or develop a group study. You can run the study in up to 10 tabs in the same browser with any worker that pleases you and all these 10 "different" workers can interact with each other. If more than 10 studies run in the same browser in parallel the oldest study is finished automatically. If you want to even more worker in parallel you can always use a different browsers: each other browser adds 10 new possible parallel-running workers.

Imitate a run from Mechanical Turk

You should always test your study before posting it anywhere. Testing that your study runs via a simple link is easy: just generate the link, start the study and run once through it. Testing studies posted in MTurk is especially cumbersome, because you should make sure that the confirmation codes are correctly displayed when the study is over. The standard way to test this is to create a study in MTurk's Sandbox. JATOS offers a way to emulate MTurk, without having to set up anything in the sandbox. Here's how.

If you think about it, MTurk simply calls a JATOS URL. The URL to start a study is normally `http://your-jatos-server/publix/study-id/start` (where `study-id` is a placeholder for the ID of the study you want to run). Two additional variables in the URL's query string tell JATOS that this request comes from MTurk (and that it should display the confirmation code when the study is over): `workerId` and `assignmentId`. Both pieces of information are normally generated by MTurk; but they can be any arbitrary string. The only constraint is that the `workerId` does not already exist within JATOS. (Think of it this way: Because a MTurk worker can run a study only once, the same `workerId` can be used only once in JATOS.)

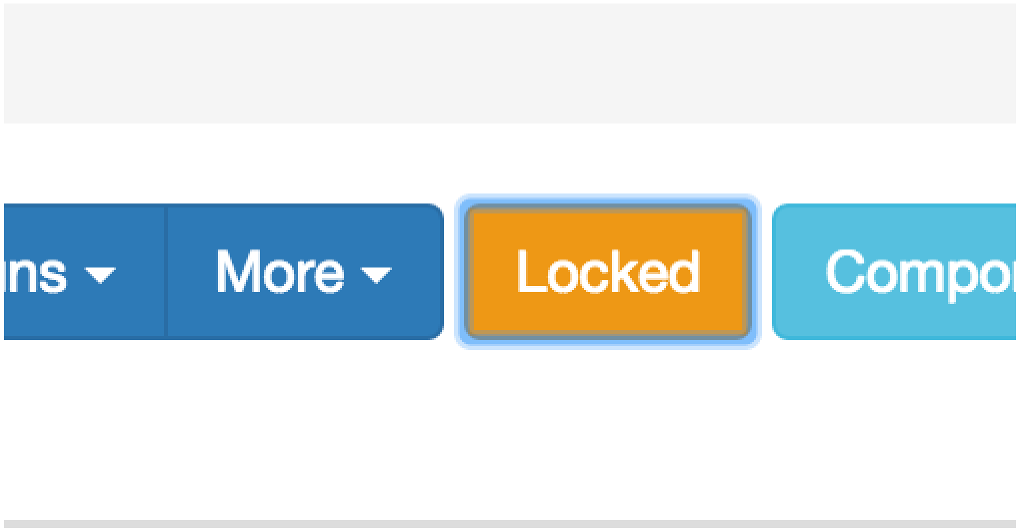
Here are some concrete examples:

To run the study with ID 4 and batch with ID 2 with an **MTurk** worker on a local JATOS use `http://localhost:9000/publix/4/start?batchId=2&workerId=123456&assignmentId=abcdef`. You can use any arbitrary value in the query strings `workerId` and `assignmentId` (in this example, `workerId = 12345` and `assignmentId = abcdef`).

To imitate a run from **MTurk's Sandbox** additionally set `turkSubmitTo` to the value 'sandbox' `http://localhost:9000/publix/4/start?batchId=2&workerId=123456&assignmentId=abcdef&turkSubmitTo=sandbox`

Lock your studies before running them

Each Study bar has a button that toggles between the 'Unlocked' and 'Locked' states. Locking a study prevents changes to its (or any of its components') properties, change the order of components, etc.



Do a General Single Run more than once in the same browser

The problem here is that a General Single Run is intended to work only once in the same browser. Although this is a feature to limit participants doing the same study twice, it can be a hassle for you as a study developer who just want to try out the General Single Run a second time. Luckily there is an easy way around: Since for a General Single Run all studies that the worker already participated in are stored in a browser cookie, it can be easily removed. Just **remove the cookie with the name JATOS_GENERALSINGLE_UUIDS** in your browser. You can find this cookie in every webpage hosted by a JATOS server. If it doesn't exist you probably never did a General Single run yet.

Continue an abandoned study

Sometimes workers leave in the middle of a study. Maybe their internet connection was down, maybe they just left for the next pub and closed the browser tab. Suppose they now want to continue from where they left it. Using the initial run link will not do what they want: it will either start a new study run, or give an error message, depending on the worker type.

But there is a way (you'll need to send the worker a new link).

You'll need three IDs: 1) *study ID*, 2) *component ID* of the component from where to restart, and 3) the *study result ID*. All three IDs are quite easy to get from JATOS' GUI. The component ID can be found in the component table of the study. The study result ID is shown in the study result table. The study ID is part of the URL of every study view, e.g. if the URL of the study view is https://cortex.jatos.org/jatos/19 then the study ID is 19.

Then the worker who abandoned the study can continue it with the link: https://my-domain-name/publix/<*study ID*>/<*component ID*>/start?srid=<*study result ID*>.

E.g.

- study ID: `31`
- component ID: `167`
- study result ID: `816`
- domain name and protocol is `https://cortex.jatos.org`

  Then the URL is: `https://cortex.jatos.org/publix/31/167/start?srid=816`

But there is a catch! This works only under three conditions:

1. the component is set to 'reloadable'
2. the worker uses the same browser on the same computer and didn't delete JATOS' cookies
3. the worker didn't start more than 10 JATOS studies at the same time in parallel after running the abandoned study

Condition 3 is very unlikely a problem and for 1 you can just check the 'reloadable' checkbox in the component's settings. Condition 2 is more difficult, it demands the worker to return to the computer and browser they run the study before.

Abort study and keep a message

If the `jatos.abortStudy` function is called (usually after the worker clicks a "Cancel" button) all result data that had been sent to JATOS during this study run will be deleted. This includes result data from prior components of the study run. But sometimes you'll want to save a bit of information that should not be deleted: you might need the worker's email address to pay them -even if they cancelled the study-. So you need a way to delete the result data but keep their email.

To do this, you can send a message together with `jatos.abortStudy` as a parameter. This message won't be deleted together with the other result data. E.g. `jatos.abortStudy("participants ID is 12345678");`. This message can then be seen in every Study Result page in the 'Message' column.

How to let a Personal Single Worker redo his study?

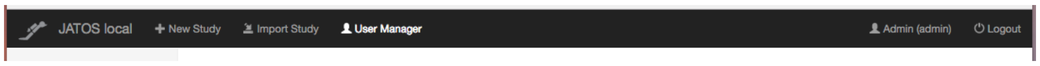
A Personal Single Worker is only allowed to run their study once. But sometimes you want to allow them to do it a second time (maybe they accidentally clicked the 'Cancel' button). One way would be to just create another Personal Single Link and hand it to the worker. But there is another way without creating a second Link: you can simply delete the worker's result from one of the result pages. This will allow this Personal Single worker to redo this study.

Manage JATOS users

Each experimenter with access to the JATOS server (though the GUI) is a JATOS User. Users can create, modify and delete the studies they are members of. They can also export and delete results. Users may also have **admin rights**, which lets them control other users' access to JATOS.

Manage users

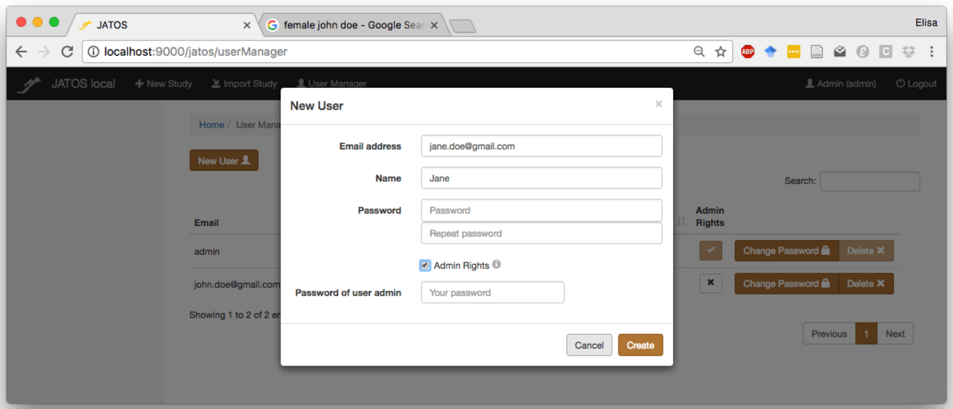
Only users with admin rights have access to the **User Manager** (located in the header on every GUI page). In the User Manager an admin can create new user, change password of other users, or delete other users.



JATOS comes with one user out-of-box: **Admin** (with user name 'admin'). Admin always has admin rights that cannot be revoked. The initial password for Admin is 'admin' should be changed immediately after installation and kept safe!

Additional to the Admin user every other user can be granted admin rights too (although they can be revoked later).

New users can be granted admin rights, by checking the corresponding box.



New User

Email address: jane.doe@gmail.com

Name: Jane

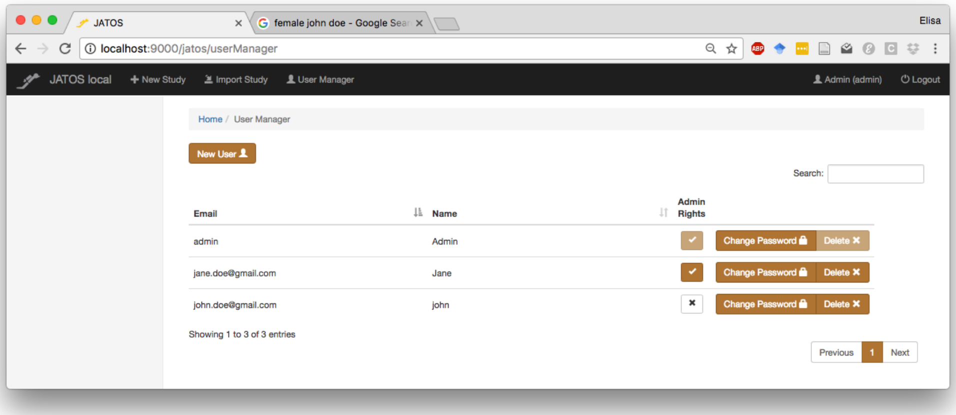
Password: Password

Repeat password

[✓] Admin Rights

Password of user admin: Your password

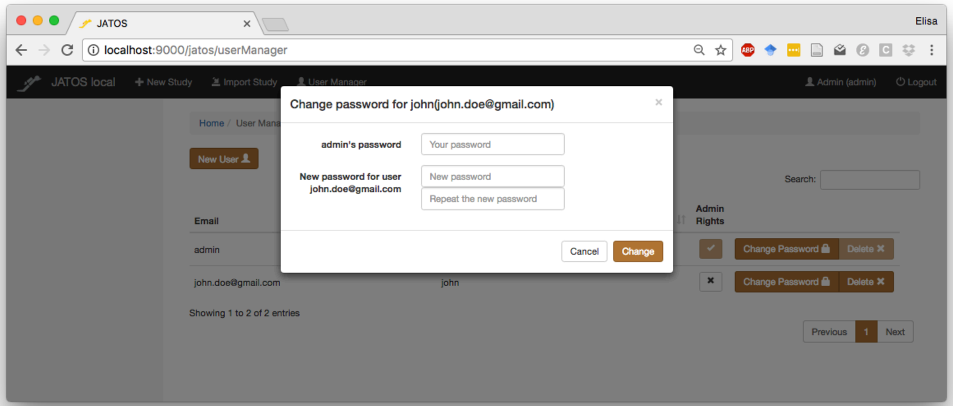
Admins can see a list of users, to grant or revoke admin rights and to delete them if necessary. **Be careful when deleting users! This will delete all studies, along with their result data, that this user is the single member of.**



Email	Name	Admin Rights		
admin	Admin	✓	Change Password	Delete ✖
jane.doe@gmail.com	Jane	✓	Change Password	Delete ✖
john.doe@gmail.com	john	✖	Change Password	Delete ✖

Showing 1 to 3 of 3 entries

Finally, admins can also change the password of other users. To change the password you'll need to enter your own (admin) password, along with the new desired password for the user.



Change password for john(john.doe@gmail.com)

admin's password	Your password
New password for user john.doe@gmail.com	New password
	Repeat the new password

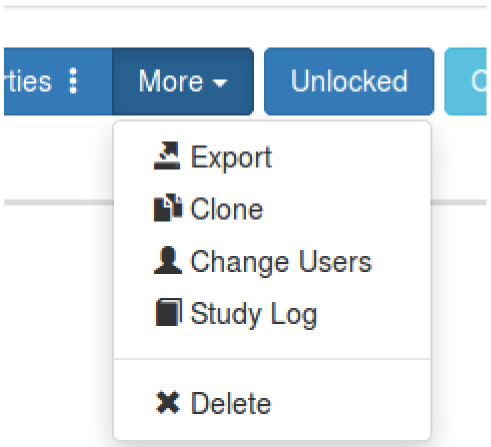
Cancel Change

Change study's members

Each experimenter with access to the JATOS server (though the GUI) is a JATOS User. Users can create, modify and delete the studies they are members of. They can also export and delete results.

A study in JATOS is allowed to have more than one users, also called members. Each member has the same rights, e.g. can run the study, create new Workers, add/change/delete components, export/delete results. Especially each member can add new members or remove existing members.

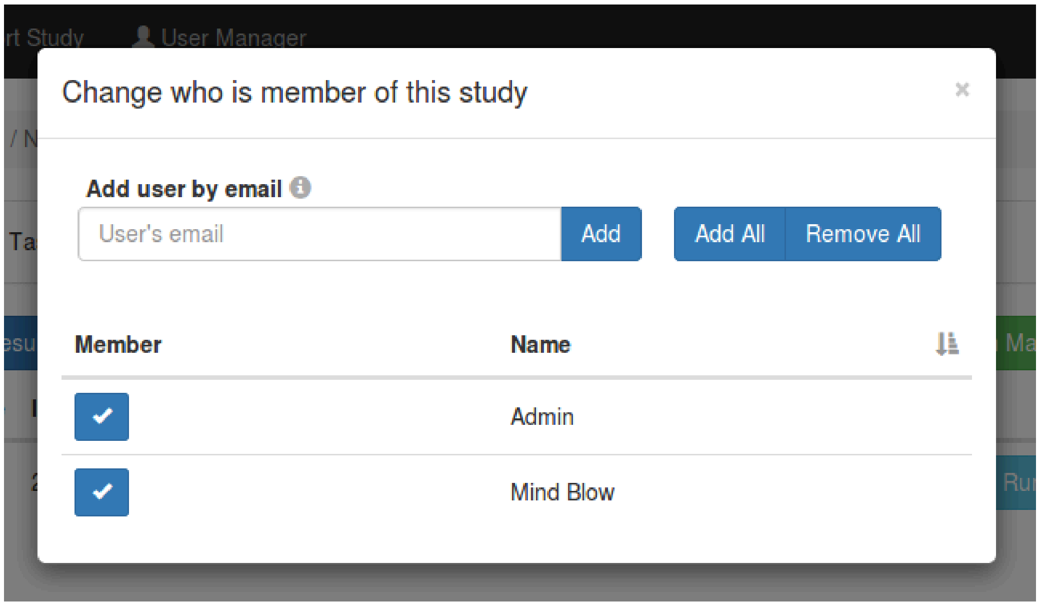
Each study has a **Change Users** button in its study toolbar.



In this menu you can add single users by their email address. Of course this works only if this is already a JATOS user. For privacy reasons JATOS never shows the email address in the member list.

A single user is removed by unchecking the checkbox in front of its name.

Additionally it is possible to add all JATOS users at once or remove all members at once.



Change who is member of this study ×

Add user by email

User's email | Add | Add All | Remove All

Member	Name
✓	Admin
✓	Mind Blow

Update JATOS

For the migration from JATOS 2 to 3 (page 12) we have an extra page with additional information about changes in jatos.js and maybe the JavaScript of your studies.

We'll periodically update JATOS with new features and bug fixes. We recommend you stay up to date with the latest release. However if you are currently running a study it's always safest to keep the same JATOS version throughout the whole experiment.

Updating a local installation of JATOS

(The procedure is different if you want to update JATOS on a server installation (page 95))

To be absolutely safe you can install the new JATOS version and keep the old one untouched. This way you can switch back if something fails. Just remember that only one JATOS can run at the same time. Always end JATOS before starting another one.

After updating you can check the new JATOS with the build-in test page localhost:9000/jatos/test in the browser. All tests should be OK.

You can update your local JATOS instance in two main ways:

First, easy way: discarding your result data

If you don't care about result data stored in JATOS:

1. Export any studies you wish to keep from the old JATOS installation.
2. Download and install the new version as if it were a new fresh download. Don't start it yet.
3. Stop the old JATOS and start the new JATOS.
4. Import all the studies your previously exported. This will transfer the files and subfolders in your study's asset folder (HTML, JavaScript, CSS files).

What will be transferred:

1. Files and subfolders in study's assets folder
2. All your studies' and components' properties
3. The **properties** of the first (Default) batch

What will be lost:

1. **All result data will be lost**
2. All workers in all batches (including Default batch)
3. All batches other than the Default batch
4. All study logs

Second way: keeping everything (including your result data)

If you do want to keep your studies, batches, and your result data you'll have to move them to the new JATOS.

1. Stop JATOS (on Unix systems, type `$ ./loader.sh stop` on the terminal. On Windows MS, close your command window)
2. Go to the folder of your old JATOS installation. From there copy your assets root folder to the new JATOS installation (Note: By default your assets root folder is called `study_assets_root` and lays in the JATOS folder but you might have changed this. You can find the location and name in `conf/production.conf`. It is specified in the line beginning with `jatos.studyAssetsRootPath=`.)
3. From the folder of your old JATOS installation copy the folders `database` and `study_logs` to the folder of the new JATOS installation.
4. If you had changed the `conf/production.conf` file in your old JATOS instance (for example to set a custom location for your `study_assets_root` folder) you'll have to do this again in the new JATOS version. We recommend re-editing the new version of the file, rather than just overwriting the new with the old version, in case anything in the `production.conf` file has changed.
5. Start the new JATOS (on Unix systems, type `$ ./loader.sh start` on the terminal. On Windows double click the `loader.bat`)
6. Open JATOS' test page in a browser `/jatos/test` and test that everything is **OK**

What will be transferred:

1. Files and subfolders in study assets folder
2. All your study and components properties
3. All batches, together with their workers, generated links, and results
4. All study logs

What will be lost: nothing

Troubleshooting

JATOS test page

JATOS comes with a test page `/jatos/test` that shows some of the currently used configuration, system properties, and does some tests, e.g. WebSockets connections and database connection. All tests should show an 'OK'.

Downloading a study / exporting a study fails (e.g. in Safari browsers)

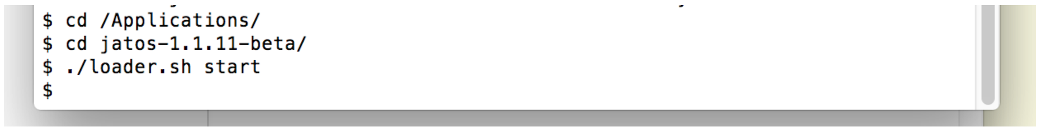
As a default, Safari (and some other browsers) automatically unzips every archive file after downloading it. When you export a study, JATOS zips your study together (study properties, all components, and all files like HTML, JavaScripts, images) and delivers it to your browser, who should save it in your local computer. Safari's default unzipping interferes with this. Follow these instructions to prevent Safari's automatic unzip.

JATOS fails to start?

(Or, if you are running Windows, do you get the message 'JATOS is already running. Press any key to continue'…)

This will happen if your computer crashed before you had the chance to close JATOS.

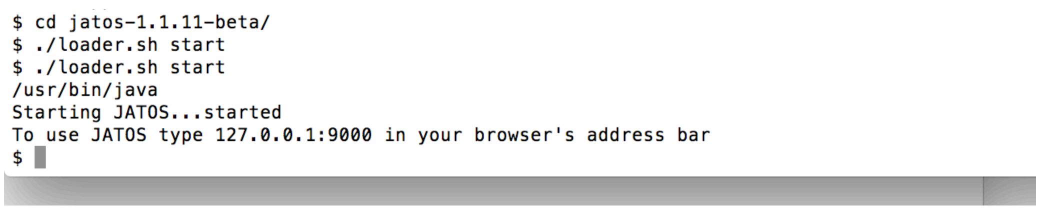
This is what you might see on a Mac Terminal if JATOS doesn't start:



```
$ cd /Applications/
$ cd jatos-1.1.11-beta/
$ ./loader.sh start
$
```

Close any open command prompt windows. Then look into your JATOS folder, and check if there's a file called `RUNNING_PID`. Delete this file and try to start JATOS again.

Here is how it should look if JATOS started successfully:



```
$ cd jatos-1.1.11-beta/
$ ./loader.sh start
$ ./loader.sh start
/usr/bin/java
Starting JATOS...started
To use JATOS type 127.0.0.1:9000 in your browser's address bar
$
```

Read log file in the browser

In a perfect world, JATOS always works smoothly and, when it doesn't, it describes the problem in an error message. Unfortunately we aren't in a perfect world: every now and then something will go wrong and you might not get any clear error messages, or no message at all. In these (rare) cases, you can look into JATOS' log file (not to be confused with the study log) to try to find what the problem might be.

The standard way to read the log file is directly on the server. You'll find your complete log file in `jatos_directory/logs/application.log`. Because JATOS is designed to avoid the command line interface, we offer a way to view your log file directly in your browser.

Just open the URL `http://your-jatos-server/jatos/log`. For privacy and security reasons, you must be logged in as an **admin** user. For example, if you're running JATOS locally with the standard settings:

http://localhost:9000/jatos/log

By default, JATOS will display the last 1000 lines of the application.log file. If you want to see more than the last 1000 lines, add the query parameter `limit`. E.g. to display the last 10000 lines on a local JATOS instance:

http://localhost:9000/jatos/log?limit=10000

A file (library, image, …) included in the HTML fails to load?

There is a common mistake Windows users make that might prevent files in the HTML from loading: Any URL or file path in a HTML file should only use '/' as a file path separator - even on Windows systems. So it should always be e.g. `<script src="/study_assets/mystudy/jsPsych-5.0.3/myscript.js"></script>` and **not** `<script src="\study_assets\mystudy\jsPsych-5.0.3\myscript.js"></script>`. And since version 3.2.3 you can leave out the path's first part and just write `<script src="myscript.js"></script>`.

Database is corrupted?

If you get an error that reads something like: `Error in custom provider, Configuration error: Configuration error[Cannot connect to database [default]]`, your database might be corrupted. By default JATOS comes with an H2 database and the H2 database doesn't handle copying its files while running too well.

There are two reasons why this might be the case: you moved your JATOS folder while it was running or you installed JATOS in a synced folder. To prevent this, be sure to always be careful with the following:

1. **Don't copy or move while JATOS is running** - Always **stop JATOS** (type `/loader.sh stop` in your Linux / Mac OS X terminal or close the window on Windows) before moving it.
2. **Don't sync while JATOS is running** - As we mentioned in the Installation page (page 5), you can run JATOS from pretty much anywhere **except** from a folder that syncs across devices, like Dropbox or Google Drive. Doing so might lead to database corruption, because while the files might be synced between computers, the running processes aren't. This will lead to havoc and destruction and, in extreme cases, to the implosion of the known Universe. You can find in our blog post a description of an attempt to recover a corrupted database. Didn't work.

Of course, this brings us to an important point: back up your result data (i.e., simply download and save your text files) regularly if you're running a study!

Example Group Studies

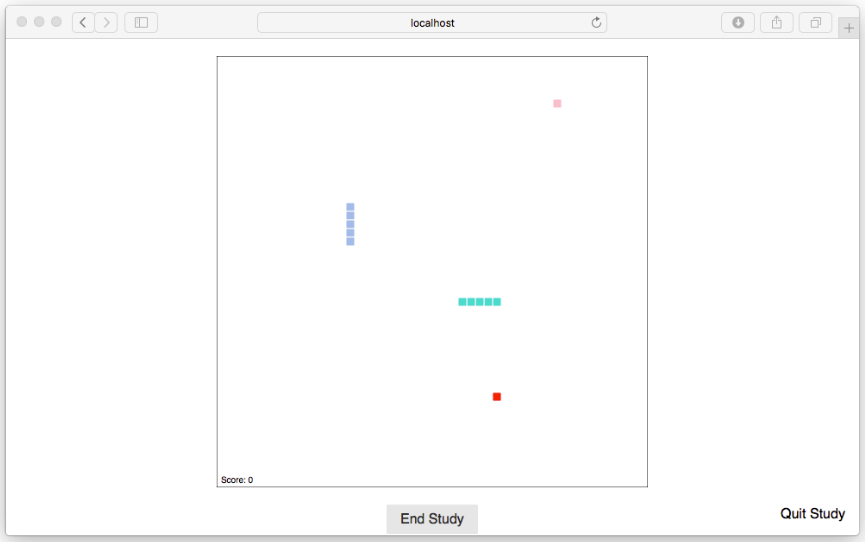
In group studies, the workers that are part of a group can communicate with each other. JATOS supports different kinds of groups. A group can e.g. have a fixed set of workers like this Prisoner's Dilemma (page 0) where exactly two workers play with each other. On the other side of the spectrum is this Snake game (page 0) with an on open, multi-worker approach.

How can you try-out a group-study if you're alone but want to simulate multiple workers?

JATOS allows up to 10 study runs (page 52) at the same time in the same browser. So you can just start the same (group) study multiple times in your browser and pretend you're many workers.

As an example of this, let's go through the Snake Game group study in detail:

1. Download and import the Snake game (page 0)
2. Open the Worker & Batch Manager (page 33)
3. Expand the "Default Batch" ("❯" button in the left) to see the worker setup
4. Now get your first worker: Expand (again with "❯") the Jatos Worker and click the **Run** button - and the study will start in a new browser tab
5. Repeat for the second worker
6. In both tabs: click through the introduction until you arrive in the waiting room. Click **Join** and then **Ready**.
7. Voilà! You'll see two snakes moving around: each tab represents one worker who is running the Snake Game - but they are in the same group
8. Optional: Add more snakes by adding more workers. You can try every worker type you want - it's of course not limited to Jatos Workers.
9. Optional: Have a look at your Group in the Worker & and Batch Manager (page 37) add see who the member workers are.



There’s actually a lot going on behind the curtain of a group study. All members of a group are able to communicate in real-time with each other directly or broadcast messages to the whole group.

Next step: Write Your Own Group Studies (page 67).

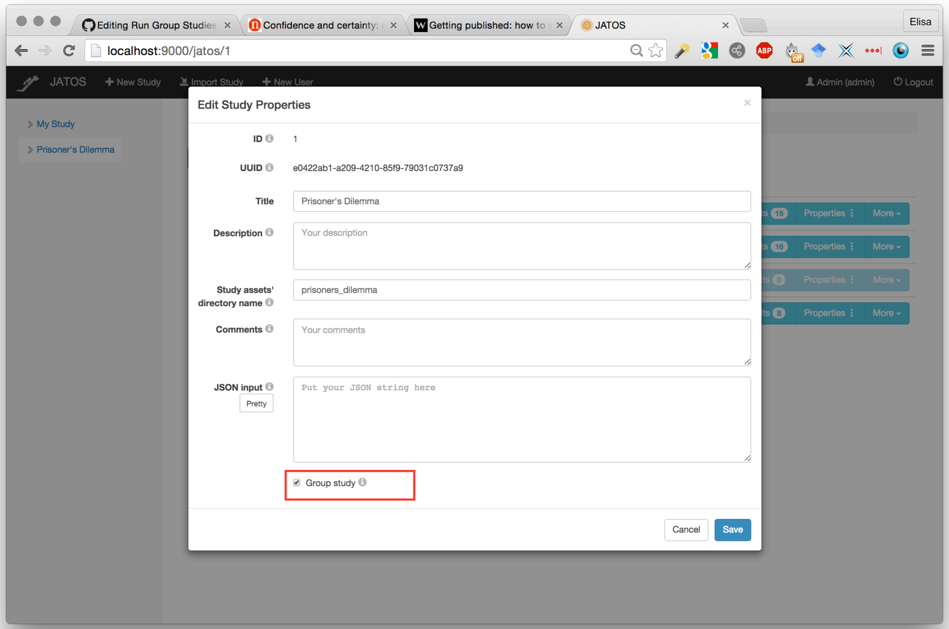
Write Group Studies I - Setup

Summary: Lern how to set up a group study and the different ways in how to assign workers to groups.

(If you haven't already, we recommend that you try out some example group studies (page 65).)

Set up group studies

First and common to all group setups is to check the Group study checkbox in the study properties.



Edit Study Properties

ID	1
UUID	e0422ab1-a209-4210-85f9-79031c0737a9
Title	Prisoner's Dilemma
Description	Your description
Study assets' directory name	prisoners_dilemma
Comments	Your comments
JSON input	Put your JSON string here

[✓ Group study]

If the Group property is checked, JATOS will assign workers into groups. We'll describe some group properties that you can use to tweak according to whether you want to keep control over worker assignment, or you give JATOS full control.

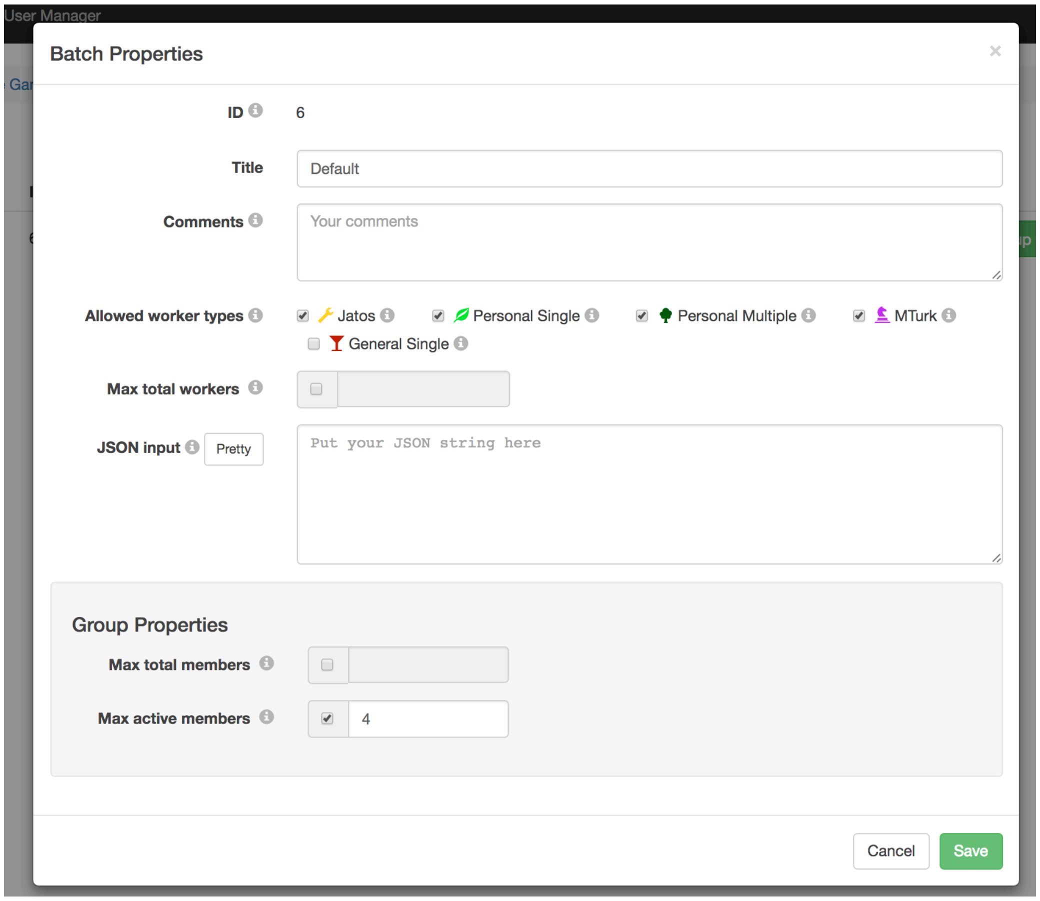
Group settings in each batch's properties

You can have multiple batches in JATOS, each one with different group settings. There are three important bits of information for a group study:

1. **Max total workers**: This isn't just a properties of group studies but can be used in single-worker studies too. It simply limits the total amount of workers who are allowed to run in this batch

2. **Max total members**: This limits the number of members a single group can have. While there can be multiple groups in a batch, the *Max total members* field applies to each separate group.
3. **Max active members**: This limits the number of active members a single group can have. An active member is in the group at this time - in opposite to a past member who already left the group. This number applies to each group separately. Example: In the Prisoner's Dilemma study, you would limit the active members to 2.

By default, all properties have no upper limit.


Batch Properties

ID	6
Title	Default
Comments	Your comments
Allowed worker types	[✓ Jatos] [✓ Personal Single] [✓ Personal Multiple] [✓ MTurk] General Single
Max total workers	
JSON input	Put your JSON string here

Group Properties

Max total members	
Max active members	[✓] 4

Group assignment

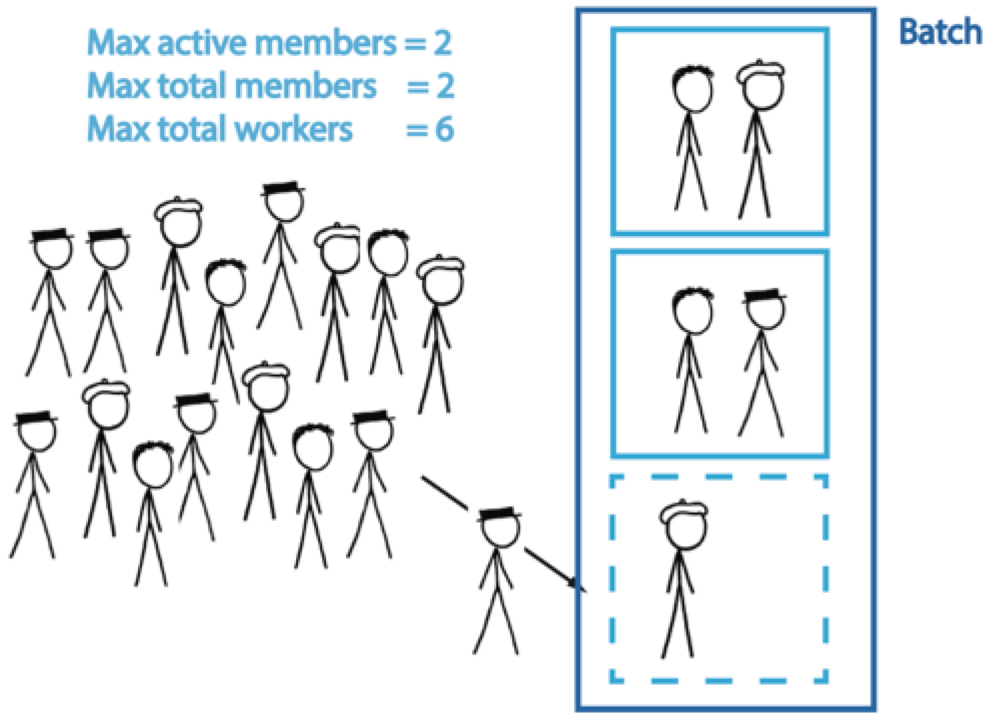
You can either tell JATOS to assign workers to different groups, or you can keep full control and do it yourself (or something in between). We'll use some example scenarios to explain how this assignment works.

Scenario 1: One group, assign workers manually

If in a batch you set the *Max total worker* to 2 and leave the other two Max parameters empty, JATOS has no other choice than to allow only 2 workers and sort them into the same group. If you then define two Personal Single workers and send the access links (displayed in the batch) to your two participants, you can be sure that they will interact with each other. If you need more groups, just create a second batch with two other workers.

Scenario 2: Several groups, let JATOS assign workers

Say you want to have 3 groups with 2 workers each. You want to leave it to JATOS which workers are paired together. Then, set *Max total workers* to 6 and both *Max active members* and *Max total members* to 2 (remember that these numbers apply to each group separately). Create your 6 workers in the Worker Setup (or use a General Single link) and distribute your link(s) to your workers.



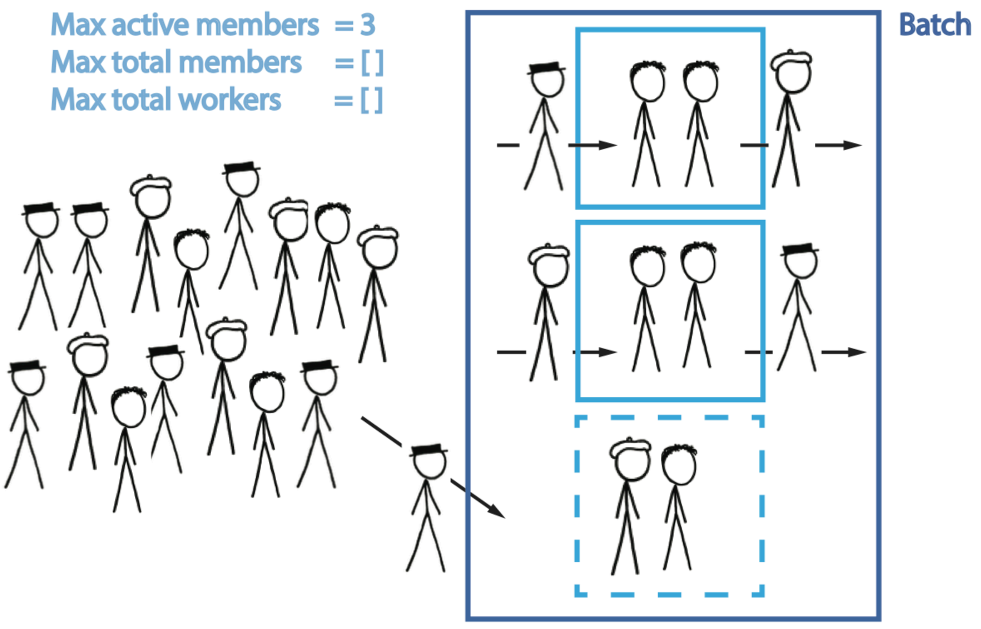
The first two scenarios may apply to the Prisoner’s Dilemma Example Study (page 0).

Scenario 3: One open world

This scenario is basically the opposite of the first one. By limiting neither the *Max total worker* nor the *Max total members*, nor the *Max active members* JATOS will sort all workers into one single group that is potentially of unlimited size. Now –to keep it completely open– just create one General Single worker and publish its link (e.g. via a mailing list or on a website). But keep in mind: this way many workers might access your study at the same time and this might overload your JATOS server.

Scenario 4: Multiple open worlds with limited active members

Say you want to have groups with up to 3 members, interacting *at the same time*. But you don't want to actually limit the total number of members per group: you want to allow new workers to join a group if one of its members left. This way each group can have a flow of workers joining and leaving - the only constraint is the maximum members per group at any given time. You also want to let JATOS set the number of groups depending on the available workers. To set up this just use one batch, set the *Max active members* to 3, and leave *Max total worker* and *Max total members* unlimited.



(Continue with Write Group Studies II - JavaScript and Messaging (page 71))

Write Group Studies II - JavaScript and Messaging

Summary: Learn how to use jatos.js to join groups and manage them. Learn about the three different ways of sending messages within a group.

Writing JavaScripts for group studies

Group studies differ from single-worker studies simply in that the JavaScript needs to handle groups and communications between members. The jatos.js library provides some useful functions for this.

If you like to dive right into jatos.js' reference:

- jatos.js functions for group studies (page 126)
- jatos.js group variables (page 103)
- jatos.js Group Session functions (page 132)

Joining a group and opening group channels

There are two requisites for allowing group members to interact:

1. Workers can only communicate with members of their own group. So, interacting workers must all join the same group. **A worker will remain in a group until jatos.js is explicitly told to leave the group (or the study run is finished). This means that if a worker moves between components or reloads a page they will remain in the same group.** This feature makes groups much more robust.
2. Communication can only be done if a group channel is open. Although jatos.js opens and closes all necessary group channels so you don't have to care for them directly while writing components. Group channels in turn use WebSockets. WebSockets are supported by all modern browsers.

So here's how a typical JATOS group study run would look like:

Component 1

- *jatos.joinGroup* -> joins group and opens group channel
- *jatos.nextComponent* -> closes group channel and jumps to next component

Component 2

- *jatos.joinGroup* -> opens group channel in the **same group**
- *jatos.nextComponent* -> closes group channel and jumps to next component

Component 3

- *jatos.joinGroup* -> opens group channel **same group**
- *jatos.endStudy* -> closes group channel, leaves group, ends component, and ends study

Notice that by calling *jatos.joinGroup (page 126)* in the second and third component JATOS does not let workers join a new group but just opens a group channel in the already joined group. To make a worker leave a group, use the function *jatos.leaveGroup* (page 129).

Every know and then you probably would like to know who the members of your groups are. This and other stats you can get by clicking on your batch's **Groups** button in the Worker & Batch Manger (page 0).

Reassigning to a different group

To move a worker from one group to a different one, use *jatos.reassignGroup* (page 130). This function will make a worker leave their group and join a different one. JATOS can only reassign to a different group if there is another group available. If there is no other group JATOS will not start a new one but put the worker into the same old group again.

Fixing a group

Sometimes you want to stay with the group like it is in the moment and don't let new members join - although it would be allowed according to the group properties. For example in the Prisoner's Example study (page 0) after the group is assembled in the waiting room component it is necessary to keep the two members as it is. Even if one of the members leaves in the middle of the game, JATOS shouldn't just assign a new member. To do this call jatos.js' function *jatos.setGroupFixed* (page 130).

Communication between group members

JATOS provides three ways for communicating within the group: direct messaging, broadcast messaging and via the Group Session.

Direct messaging

Members can send direct messages to a single other member of the same group with the *jatos.sendGroupMsgTo* (page 128) function. Like broadcast messaging this way of group communication is fast but can be unreliable in case of an unstable network connection. We use direct messaging in the Snake example (page 0) to send the coordinates of the snakes on every step. Here, speed is more critical than reliability in the messages, because a few dropped frames will probably go unnoticed.

Broadcast messaging

Members can send messages to all other members of the same group with the *jatos.sendGroupMsg* (page 128) function. Like direct messaging this way of group communication is fast but can be unreliable in case of an unstable network connection.

Group session

The Group Session is one of the three types of session that JATOS provides (page 28). Members can access the Group Session data with the Group Session functions (page 132). The Group Session data are stored in JATOS' database **only while the group is active. It is deleted when the group is finished.** Communication via Group Session is slower, but more reliable than group messaging. If one member has an unstable internet connection or does a page reload, the Group Session will be automatically restored after the member reopens the group channel. Workers communicate via the Group Session data in the Prisoner's Example study (page 0), because here one dropped message would lead to important information loss.

Install JATOS on a server

> **Summary:** To run studies online, e.g. with Mechanical Turk, JATOS has to be installed on a server. Server instances of JATOS have slightly different configuration requirements than local instances. This text aims at server admins who wants to setup a server running JATOS and who know their way around server management.

There are several ways to bring JATOS to the internet. You can install it

- on your own dedicated server
- in the cloud (with IaaS)
- in the cloud with a Docker container

The first two are discussed here in this page. The last one has its own page (page 97).

One word about IaaS. There are many IaaS providers (Digital Ocean, Microsoft Azure, Google Cloud, Amazon's AWS etc.). They all give you some kind of virtual machine (VM) and the possibility to install an operating system on it. I'd recommend to go with a Linux system like Ubuntu or Debian. Another point is to make sure they have persistent storage and not what is often called 'ephemeral storage' (storage that is deleted after the VM shuts down). Since JATOS stores all study assets in the server's file system persistent storage is needed. But apart from that it's the same JATOS installation like on a dedicated server. In JATOS in Amazon's Cloud (without Docker) (page 99) we have some advice in how to do it in AWS.

The actual JATOS instance on a server isn't too different from a local one. It basically involves telling JATOS which IP address and port it should use and (optionally) replace the H2 database with a MySQL one. There are other issues however, not directly related to JATOS, that you should consider when setting up a server. These include: setting up automatic, regular backups of your data, an automatic restart of JATOS after a server reboot, encryption, additional HTTP server, etc.

Installation on a server

1. Install Java

We've produced multiple versions of JATOS. The simplest version is JATOS alone, but other versions are bundled with Java JRE. On a server, it's best (though not necessary) to install JATOS without a bundled Java. This will make it easier to upgrade to new Java releases.

2. Install JATOS

1. Download JATOS

   E.g. with *wget* for the version 3.2.1:

   `wget https://github.com/JATOS/JATOS/releases/download/v3.2.1/jatos-3.2.1.zip`

2. JATOS comes zipped. Unpack this file at a location in your server's file system where JATOS should be installed.
3. Check that the file `loader.sh` in the JATOS folder is executable.
4. Check that JATOS starts with `loader.sh start|restart|stop`

3. Configuration

If JATOS runs locally it's usually not necessary to change the defaults but on a server you probably want to set up the IP and port or maybe use a different database and change the path of the study assets root folder. These docs have an extra page on how to Configure JATOS on a Server (page 80).

4. Change Admin's password

Every JATOS installation comes with an Admin user that has the default password 'admin'. You must change it before the server goes live. This can be done in JATOS' GUI:

1. Start JATOS and in a browser go to JATOS login page `http://your-domain-or-IP/jatos`
2. Login as 'admin' with password 'admin'
3. Click on 'Admin (admin) in top-right header
4. Click 'Change Password'

5. Check JATOS' test page

JATOS comes with a handy test page `http://your-domain-or-IP/jatos/test`. It shows some of the current configuration and system properties. Above all it does some tests, e.g. WebSockets connections and database connection. Check that all tests show an 'OK'.

6. [Optional] HTTP server and encryption

Most admins tend to use an additional HTTP server in front of JATOS for encryption purpose. We provide two example configurations for Nginx and Apache. Both support encryption and WebSockets (keep in mind JATOS relies on WebSockets and it's necessary to support them).

- JATOS with Nginx (page 85)
- JATOS with Apache (page 93)

7. [Optional] Auto-start JATOS

It's nice to have JATOS starts automatically after a start or a reboot of your machine. Choose between one of the two possibilities: 1) via a systemd service (JATOS version >= 3.1.6, recommended), or 2) via a init.d script.

1) Via systemd service (JATOS version >= 3.1.6, recommended)

Create a systemd service file for JATOS

```
vim /etc/systemd/system/jatos.service
```

and put the following text inside.

```
[Unit]
Description=JATOS
After=network-online.target
# If you use JATOS with an MySQL database use
#After=network-online.target mysql.service

[Service]
Type=forking
PIDFile=/my/path/to/jatos/RUNNING_PID
User=my-jatos-user
ExecStart=/my/path/to/jatos/loader.sh start
ExecStop=/bin/kill $MAINPID
ExecStopPost=/bin/rm -f /my/path/to/jatos/RUNNING_PID
ExecRestart=/bin/kill $MAINPID
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
```

Change the paths and the user according to your JATOS path and the user you want to start JATOS with.

Secondly, notify systemd of the new service file:

```
systemctl daemon-reload
```

and enable it, so it runs on boot:

```
systemctl enable jatos.service
```

That's it.

Additionally you can manually start/stop JATOS now with:

- `systemctl start jatos.service`
- `systemctl stop jatos.service`
- `systemctl restart jatos.service`
- `systemctl status jatos.service`

You can disable the service with `systemctl disable jatos.service`. If you change the service file you need `systemctl daemon-reload jatos.service` to let the system know.

2) Via /etc/init.d script

It's easy to turn the `loader.sh` script into an init script for a daemon.

1. Copy the `loader.sh` script to `/etc/init.d/`
2. Rename it to `jatos`
3. Change access permission with `chmod og+x jatos`
4. Edit `/etc/init.d/jatos`
    a. Comment out the line that defines the JATOS location `dir="$( cd "$( dirname "$0" )" && pwd )"`
    b. Add a new locatoin `dir=` with the path to your JATOS installation

    The beginning of your `/etc/init.d/jatos` should look like:

    ```
    #!/bin/bash
    # JATOS loader for Linux and MacOS X

    # Change IP address and port here
    # Alternatively you can use command-line argument
    s -Dhttp.address and -Dhttp.port
    address="127.0.0.1"
    port="9000"

    # Don't change after here unless you know what yo
    u're doing
    ##################################################
    ##########

    # Get JATOS directory
    #dir="$( cd "$( dirname "$0" )" && pwd )"
    dir="/path/to/my/JATOS/installation"
    ...
    ```

5. Make it auto-start with the command `sudo update-rc.d jatos defaults`

Now JATOS starts automatically when you start your server and stops when you shut it down. You can also use the init script yourself like any other init script with `sudo /etc/init.d/jatos start|stop|restart`.

8. [Optional] Backup

The easiest way to backup is to let JATOS users care themselves for their own data. JATOS has an easy to use export function for result data (page 0). So you could just tell everyone to export their data regularily.

But if you want to set up a regular backup of the data stored in JATOS here are the necessary steps. Those data consists of two parts (1.) the data stored in the database and (2.) your study assets folder that contains all the web files (e.g. HTML, JavaScript, images). Both have to be backed up to be able to restore them later on.

1. **Database**

    - **MySQL** - If you use a MySQL database you might want to look into the `mysqldump` shell command. E.g., with `mysqldump -u myusername -p mydbname > mysql_bkp.out` you can backup the whole data into a single file. Restore the database with `mysql -u myusername -p mydbname < mysql_bkp.out`.
    - **H2** - There are at least two ways: one easy (but unofficial) and one official:

        1. Copy & paste the db file - It’s important to **stop JATOS** before doing a backup or restoring a H2 database this way. If you do not stop JATOS your data might be corrupted (page 63). You can just backup the folder `my-jatos-path/database`. In case you want to restore an older version from the backup just replace the current version of the folder with the backed-up version.
        2. Via H2’s upgrade, backup, and restore tool

2. **Study assets root folder** - You can just make a backup of your study assets folder. If you want to return to a prior version replace the current folder with the backed-up version.

Remember, a backup has to be done of **both** the *database* **and** the *study assets root folder*.

Configure JATOS on a Server

> **Summary:** If JATOS runs locally it's usually not necessary to change the defaults. On a server, you may want to set up the IP and port, or use a different database, change the path of the study assets root folder, or add some password restrictions.

Restart JATOS after making any changes to the configuration (`loader.sh restart`)

IP / domain and port

By default JATOS uses the address 127.0.0.1 and port 9000. There are two ways to configure the host name or IP address and the port:

1. In `loader.sh` change the values of 'address' and 'port' according to your IP address or domain name and port.

   ```
   address="172.16.0.1"
   port="8080"
   ```

2. Via command-line arguments `-Dhttp.address` and `-Dhttp.port`, e.g. with the following command you'd start JATOS with IP 10.0.0.1 and port 80

   ```
   loader.sh start -Dhttp.address=10.0.0.1 -Dhttp.port=80
   ```

URL base path (JATOS >= v3.3.1)

JATOS can be configured to use an base path. E.g we have the host "www.example.org" and let JATOS run under "mybasepath" so that JATOS' URL all start with "www.example.org/mybasepath/". This can be achieved in two ways:

1. Via the command-line argument `-DJATOS_URL_BASE_PATH`, e.g.

   ```
   loader.sh start -DJATOS_URL_BASE_PATH="/mybasepath/"
   ```

2. Via `conf/production.conf`: change `play.http.context`

```
play.http.context = "/mybasepath/"
```

The path always has to start and end with a "/". And keep in mind that if you add a base path to JATOS' URL you have to adjust all absolute paths to the study assets (in HTML and JavaScript files) too - or use relative paths (page 20).

Study assets root path

By default the study assets root folder (where all your study's HTML, JavaScript files etc. are stored) is located within JATOS installation's folder in `study_assets_root`. There are three ways to change this path:

1. Via the command-line argument `-DJATOS_STUDY_ASSETS_ROOT_PATH`, e.g.

   ```
   loader.sh start -DJATOS_STUDY_ASSETS_ROOT_PATH="/path/t
   o/my/assets/root/folder"
   ```

2. Via `conf/production.conf`: change `jatos.studyAssetsRootPath`

   ```
   jatos.studyAssetsRootPath="/path/to/my/jatos_study_asset
   s_root"
   ```

3. Via the environment variable `JATOS_STUDY_ASSETS_ROOT_PATH`, e.g. the following export adds it to the env variables:

   ```
   export JATOS_STUDY_ASSETS_ROOT_PATH="/path/to/my/assets/
   root/folder"
   ```

External Database

By default JATOS uses an embedded H2 database and no further setup is necessary but it can be easily configured to work with a MySQL database.

Possible scenarios why one would use an external database are

- the expected traffic is rather high and many users will run studies with many participants and the studies will store a lot of data in the database
- to be able to do a regular database backup

- higher trust in the reliability of a MySQL database (although we never had problems with H2)

One could install the external database on the same server as JATOS is running or on an extra server depending on ones need.

Appart from giving JATOS access to the external database it is not necessary to set it up any further, e.g. to create any tables inside database - JATOS is doing this automatically.

You can confirm that JATOS is accessing the correct database by looking in the logs. One of the lines after JATOS starts should look like this (with your JDBC URL).

```
19:03:42.000 [info] - p.a.d.DefaultDBApi - Database [default] c
onnected at jdbc:mysql://localhost/jatos?characterEncoding=UT
F-8
```

JATOS requires MySQL >= 5.5 or H2 >= 1.4.192 (prior versions might work - I just never tested)

Note: If you want to use a MySQL database consider using the *utf8mb4* Character Set with 4-Byte UTF-8 Unicode Encoding. Only this character set contains the whole unicode and you won't run into issues like this one.

There are three ways to set up JATOS to work with an external database:

1. Via command-line arguments:

    - `-DJATOS_DB_URL` - specifies the JDBC URL to the database
    - `-DJATOS_DB_USERNAME` and `-DJATOS_DB_PASSWORD` - set username and password
    - `-DJATOS_DB_DRIVER` - can be either `org.h2.Driver` or `com.mysql.jdbc.Driver`
    - `-DJATOS_JPA` - can be either `h2PersistenceUnit` or `mysqlPersistenceUnit`

    E.g. to connect to a MySQL database running on 172.17.0.2 and a table named 'jatos' use:

    ```
    loader.sh start -DJATOS_DB_URL='jdbc:mysql://172.17.0.2/
    jatos?characterEncoding=UTF-8' -DJATOS_DB_USERNAME=sa -D
    JATOS_DB_PASSWORD=sa -DJATOS_JPA=mysqlPersistenceUnit -D
    JATOS_DB_DRIVER=com.mysql.jdbc.Driver
    ```

2. Via `conf/production.conf` (description analog to 1.)

   ```
   db.default.url="jdbc:mysql://localhost/MyDatabase?charac
   terEncoding=UTF-8"
   db.default.user=myusername
   db.default.password=mypassword
   db.default.driver=com.mysql.jdbc.Driver
   jpa.default=mysqlPersistenceUnit
   ```

3. Via environment variables (description analog to 1.)

   - `JATOS_DB_URL`
   - `JATOS_DB_USERNAME`
   - `JATOS_DB_PASSWORD`
   - `JATOS_DB_DRIVER`
   - `JATOS_JPA`

   E.g. to set all database environment variables for a MySQL database and table called 'jatos' you could use a command (change the values):

   ```
   export JATOS_DB_URL='jdbc:mysql://localhost/jatos?charac
   terEncoding=UTF-8' JATOS_DB_USERNAME='jatosuser' JATOS_D
   B_PASSWORD='mypassword' JATOS_DB_DRIVER=com.mysql.jdbc.D
   river JATOS_JPA=mysqlPersistenceUnit
   ```

Password restrictions

By default JATOS' keeps it simple and relies on the users to choose save passwords: it just enforces a length of at least 7 characters. But this can be changed in the `conf/production.conf` with the following two properties.

- `jatos.user.password.length` - Set with an positive integer (default is 7)
- `jatos.user.password.strength` - Set to 0, 1, 2, or 3 (default is 0)
  - `0` : No restrictions on characters
  - `1` : At least one Latin letter and one number
  - `2` : At least one Latin letter, one number and one special character (`#?!@$%^&*-`)

  - `3`: At least one uppercase Latin letter, one lowercase Latin letter, one number and one special character (`#?!@$%^&*-`)

Other configuration in production.conf

Additional to the database (page 0) and the study assets root path (page 81) some other properties can be configured in the `conf/production.conf`.

- `jatos.userSession.timeout` - time in minutes a user stays logged in (default is 1440 = 1 day)
- `jatos.userSession.inactivity` - defines the time in minutes a user is automatically logged out after inactivity (default is 60)
- `jatos.userSession.validation` - (since JATOS >= 3.1.10) - toggles user session validation: set to false to switch it off (default is true) - WARNING: Turning off the user session validation reduces JATOS security! Sometimes this is necessary, e.g. if your users have a extremely dynamic IP address.
- `play.http.session.secure` - secure session cookie: set true to restrict user access to HTTPS (default is false)

Apart from those all configuration properties possible in the Play Framework are possible in JATOS production.conf too.

JATOS with Nginx

These are examples for configurations of Nginx as a proxy in front of JATOS. It is not necessary to run JATOS with a proxy but it's common. They support WebSockets for JATOS' group studies.

The following two configs are the content of `/etc/nginx/nginx.conf`. Change them to your needs. You probably want to change your servers address (`www.example.com` in the example) and the path to the SSL certificate and its key. Those `proxy_set_header X-Forwarded-*` and `proxy_set_header X-Real-IP` are necessary to tell JATOS the real requester's IP address - please leave them unchanged.

As an additional security measurement you can uncomment the `location /jatos` and config your local network. This will restrict the access to JATOS' GUI (every URL starting with `/jatos`) to the local network.

A JATOS server that handles sensitive or private data should always use encryption (HTTPS).

With HTTPS

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;

events {
        worker_connections 768;
        # multi_accept on;
}

http {
        sendfile on;
        keepalive_timeout 65;
        client_max_body_size 500M;

        include /etc/nginx/mime.types;
        default_type application/octet-stream;

        proxy_buffering    off;
        proxy_set_header   X-Real-IP $remote_addr;
        proxy_set_header   X-Forwarded-Proto https;
        proxy_set_header   X-Forwarded-Ssl on;
        proxy_set_header   X-Forwarded-For $proxy_add_x_forward
ed_for;
        proxy_set_header   Host $http_host;
        proxy_http_version 1.1;

        upstream jatos-backend {
                server 127.0.0.1:9000;
        }

        # needed for websockets
        map $http_upgrade $connection_upgrade {
                default upgrade;
                ''      close;
        }

        # redirect http to https
        server {
                listen       80;
                server_name www.example.com;
                rewrite ^ https://www.example.com$request_uri?
permanent;
        }

        server {
                listen                  443;
                ssl                     on;
                # http://www.selfsignedcertificate.com/ is usef
ul for development testing
                ssl_certificate      /etc/ssl/certs/localhost.c
rt;
                ssl_certificate_key  /etc/ssl/private/localhos
t.key;

                # From https://bettercrypto.org/static/applie
d-crypto-hardening.pdf
                ssl_prefer_server_ciphers on;
                ssl_protocols TLSv1 TLSv1.1 TLSv1.2; # not poss
ible to do exclusive
                ssl_ciphers 'EDH+CAMELLIA:EDH+aRSA:EECDH+aRSA+A
ESGCM:EECDH+aRSA+SHA384:EECDH+aRSA+SHA256:EECDH:+CAMELLIA256:+A
ES256:+CAMELLIA128:+AES128:+SSLv3:!aNULL:!eNULL:!LOW:!3DES:!MD
5:!EXP:!PSK:!DSS:!RC4:!SEED:!ECDSA:CAMELLIA256-SHA:AES256-SHA:C
AMELLIA128-SHA:AES128-SHA';
                add_header Strict-Transport-Security "max-age=1
5768000; includeSubDomains";

                keepalive_timeout    70;
                server_name www.example.com;

                # websocket location (JATOS' group and batch ch
annel and the test page)
                location ~ "/(jatos/testWebSocket|publi
x/[\d]+/(group/join|batch/open))" {
                        proxy_pass              http://jatos-ba
ckend;
                        proxy_http_version      1.1;
                        proxy_set_header        Upgrade $http_u
pgrade;
                        proxy_set_header        Connection $con
nection_upgrade;
                        proxy_connect_timeout   7d; # keep ope
n for 7 days even without any transmission
                        proxy_send_timeout      7d;
                        proxy_read_timeout      7d;
                }

                # restrict access to JATOS' GUI to local networ
k
                #location /jatos {
                #       allow   192.168.1.0/24;
                #       deny    all;
                #}
                # all other traffic
                location / {
                        proxy_pass              http://jatos-ba
ckend;
                }
        }

        access_log /var/log/nginx/access.log;
        error_log /var/log/nginx/error.log;

        include /etc/nginx/conf.d/*.conf;
        include /etc/nginx/sites-enabled/*;
}
```

Simple without encryption

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;

events {
        worker_connections 768;
        # multi_accept on;
}

http {
        sendfile on;
        keepalive_timeout 65;
        client_max_body_size 500M;

        include /etc/nginx/mime.types;
        default_type application/octet-stream;

        proxy_buffering    off;
        proxy_set_header   X-Real-IP $remote_addr;
        proxy_set_header   X-Forwarded-Proto http;
        proxy_set_header   X-Forwarded-Ssl on;
        proxy_set_header   X-Forwarded-For $proxy_add_x_forward
ed_for;
        proxy_set_header   Host $http_host;
        proxy_http_version 1.1;

        upstream jatos-backend {
                server 127.0.0.1:9000;
        }

        # needed for websockets
        map $http_upgrade $connection_upgrade {
                default upgrade;
                ''      close;
        }

        server {
                listen               80;

                keepalive_timeout    70;
                server_name          www.example.com;

                # websocket location (JATOS' group and batch ch
annel and the test page)
                location ~ "^/(jatos/testWebSocket|publi
x/[\d]+/(group/join|batch/open))" {
                        proxy_pass           http://jatos-ba
ckend;
                        proxy_http_version      1.1;
                        proxy_set_header        Upgrade $http_u
pgrade;
                        proxy_set_header        Connection $con
nection_upgrade;
                        proxy_connect_timeout   7d; # keep ope
n for 7 days even without any transmission
                        proxy_send_timeout      7d;
                        proxy_read_timeout      7d;
                }

                # restrict access to JATOS' GUI to local networ
k
                #location /jatos {
                #       allow   192.168.1.0/24;
                #       deny    all;
                #}

                # all other traffic
                location / {
                        proxy_pass              http://jatos-ba
ckend;
                }
        }

        access_log /var/log/nginx/access.log;
        error_log /var/log/nginx/error.log;

        include /etc/nginx/conf.d/*.conf;
        include /etc/nginx/sites-enabled/*;
}
```

JATOS with Apache

This is an example of a configuration of Apache as a proxy in front of JATOS. While it’s not necessary to run JATOS with a proxy, it’s common to do so in order to allow encryption.

Here I used Apache 2.4.18 on a Ubuntu system. I recommend to use at least **version 2.4** since JATOS relies on WebSockets that aren’t supported by earlier Apache versions.

I had to add some modules to Apache to get it working:

```
sudo a2enmod rewrite
sudo a2enmod proxy_wstunnel
sudo a2enmod proxy
sudo a2enmod headers
sudo a2enmod ssl
sudo a2enmod lbmethod_byrequests
sudo a2enmod proxy_balancer
sudo a2enmod proxy_http
sudo a2enmod remoteip
```

The following is an example of a proxy config with Apache. I stored it in `/etc/apache2/sites-available/example.com.conf` and added it to Apache with the command `sudo a2ensite example.com.conf`.

- It enforces access via HTTPS by redirecting all HTTP traffic.
- As an additional security measurement you can uncomment the `<Location "/jatos">` and config your local network. This will restrict the access to JATOS’ GUI (every URL starting with `/jatos`) to the local network.

```
<VirtualHost *:80>
  ServerName www.example.com

  # Redirect all unencrypted traffic to the respective HTTPS pa
ge
  Redirect "/" "https://www.example.com/"
</VirtualHost>

<VirtualHost *:443>
  ServerName www.example.com

  # Restrict access to JATOS GUI to local network
  #<Location "/jatos">
  #   Order deny,allow
  #   Deny from all
  #   Allow from 127.0.0.1 ::1
  #   Allow from localhost
  #   Allow from 192.168
  #</Location>

  # Needed for JATOS to get the correct host and protocol
  ProxyPreserveHost On
  RequestHeader set X-Forwarded-Proto "https"
  RequestHeader set X-Forwarded-Ssl "on"

  # Your certificate for encryption
  SSLEngine On
  SSLCertificateFile /etc/ssl/certs/localhost.crt
  SSLCertificateKeyFile /etc/ssl/private/localhost.key

  # JATOS uses WebSockets for its batch and group channels
  RewriteEngine On
  RewriteCond %{HTTP:Upgrade} =websocket [NC]
  RewriteRule /(.*)           ws://localhost:9000/$1 [P,L]
  RewriteCond %{HTTP:Upgrade} !=websocket [NC]
  RewriteRule /(.*)           http://localhost:9000/$1 [P,L]

  # Proxy everything to the JATOS running on localhost on port
9000
  ProxyPass / http://localhost:9000/
  ProxyPassReverse / http://localhost:9000/
</VirtualHost>
```

Updating a JATOS server installation

Updating the server instance is equivalent to doing it locally (page 60), but make sure that you know what you're doing; especially if you have paired JATOS with a MySQL database.

To be absolutely safe you can install the new JATOS version and keep the old one untouched. This way you can switch back if something fails. Just remember that only one JATOS can run at the same time.

Note: If you are using a MySQL database and to be on the safe side in case something goes wrong, make a backup of your MySQL database. Dump the database using `mysqldump -u yourUserName -p yourDatabaseName > yourDatabaseName.out`.

As with updating of a local JATOS installation (page 60) you have two options: 1. Keep your studies but discard all your result data and batches. 2. Keep everything, including your studies and result data (might not always be possible).

After updating you can check the new JATOS installation with the test page `my-address/jatos/test` in the browser. All tests should be OK.

First option: quick and dirty (discarding result data)

You can just follow the update instructions for the local installation (page 60). If you use a mySQL database don't forget to configure it with a clean and new one (page 80) (not the one from your old JATOS). Do not use the new JATOS with the old MySQL database unless you choose to keep your data, as described below.

Second option: keeping everything

This means that we have to configure the MySQL database or copy the H2 database files.

1. Stop the old JATOS using `./loader.sh stop`
2. Copy the new JATOS version to your server, e.g. copy it into the same folder where your old JATOS is located. Don't yet remove the old JATOS instance.
3. Unzip the new JATOS (`unzip jatos-x.x.x-beta.zip`)
4. From the old JATOS installation copy your assets root folder to the new JATOS installation (Note: By default your assets root folder is called `study_assets_root` and lays in the JATOS folder but you might have changed this (page 80).
5. Database

    - H2 - If you are using the default H2 database: From your the folder of your old JATOS installation copy the folder `database` to the new JATOS installation. Remember to stop JATOS before copying the folder (page 63).
    - MySQL - For MySQL you don't have to change anything on the database side.
6. From the old JATOS installation copy the folder `study_logs` to the new JATOS installation
7. Configure the new JATOS like the old one (page 80) - usually it's enough to copy the `production.conf` from the old `conf` folder into the new one
8. Start the new JATOS using `./loader.sh start`
9. Open JATOS' test page in a browser `/jatos/test` and test that everything is **OK**

Install JATOS via Docker

JATOS has a Docker image: hub.docker.com/r/jatos/jatos/

Docker is a great technology, but if you never heard of it you can safely ignore this page (it's not necessary to use it if you want to install JATOS, either locally or on a server).

If you have heard of it, you might want to, for example, setup JATOS in a Docker container together with an external H2 or MySQL database in another Docker container. You can easily do that:

Install JATOS locally with a Docker container

1. Install Docker locally on your computer (not covered here)
2. Go to your shell or command line interface
3. Pull the JATOS image: use `docker pull jatos/jatos:x.x.x` and specify the release, e.g. to get version 2.2.4 use `docker pull jatos/jatos:2.2.4`
4. Check that you actually downloaded the image: `docker images` should show `jatos/jatos` in one line
5. Now run JATOS (and create a new Docker container) with `docker run -d -p 9000:9000 jatos/jatos:x.x.x`, e.g. to for version 2.2.4 use `docker run -d -p 9000:9000 jatos/jatos:2.2.4`. The `-d` argument specifies to run this container as a daemon and the `-p` is responsible for the port mapping.
6. Check that the new container is running: In your browser go to localhost:9000 - it should show the JATOS login screen. Or use `docker ps -a` - in the line with `jatos/jatos` the status should say `up`.

Troubleshooting: By removing the `-d` argument (e.g. `docker run -p 9000:9000 jatos/jatos:2.2.4`) you get JATOS' logs printed in your shell - although you don't run it as a daemon in the background anymore.

Change port

With Docker you can easily change JATOS' port (actually we change the port mapping of JATOS' docker container). Just use Docker `-p` argument and specify your port. E.g. to run JATOS on port 8080 use `docker run -d -p 8080:9000 jatos/jatos:2.2.4`.

Configure with environment variables

All environment variables that can be used to configure a normal JATOS server installation (page 80) can be used in a docker installation. Just use Docker's `-e` argument to set them.

E.g. to setup JATOS with a MySQL database running under IP `172.17.0.2` that uses the table `jatos` use the following command (but change the JATOS version and use username and password of your MySQL account):

```
docker run -e JATOS_DB_URL='jdbc:mysql://172.17.0.2/jatos?chara
cterEncoding=UTF-8' -e JATOS_DB_USERNAME='root' -e JATOS_DB_PAS
SWORD='password' -e JATOS_DB_DRIVER=com.mysql.jdbc.Driver -e JA
TOS_JPA=mysqlPersistenceUnit -p 9000:9000 jatos/jatos:2.2.4
```

JATOS in Amazon's Cloud (without Docker)

On this page is additional information in how to install JATOS on a server in Amazon’s Web Services. All general installation advice is in JATOS on a server (page 74) and applies here too.

1. First you need to register at AWS (Amazon Web Services) (they’ll want your credit card).
2. In AWS webpage move to EC2 and launch a new instance with Ubuntu (you can use other Linux too - I tested it with Ubuntu 14.04 and 16.04)
3. During the creation of the new EC2 instance you will be ask whether you want to create a key pair. Do so. Download the file with the key (a *.pem file). Store it in a safe place - with this key you will access your server.
4. Login via SSH: `ssh -i /path/to/your/pem_key_file ubuntu@xx.xx.xx.xx` (Use your instance’s IP address: In AWS / EC2 / Instances / Description are two IPs ‘Private IP’ and ‘Public IP’. Use the **public** one.)
5. Get the latest JATOS bundled with Java (exchange x.x.x with the version you want) `wget https://github.com/JATOS/JATOS/releases/download/vx.x.x/jatos-x.x.x-linux_java.zip`
6. `unzip jatos-x.x.x-linux_java.zip` (You probably have to install ‘unzip’ first with `sudo apt-get install unzip`.)
7. Configure IP and port in `jatos.sh`: Use the ‘**Private IP**’ from your instance description (the one that starts with 172.x.x.x) and port 80
8. Allow inbound HTTP/HTTPS traffic: This is done in AWS GUI.
9. (Optional) auto-start JATOS (page 76)
10. Change JATOS’ admin password

jatos.js Reference

> **Summary:** jatos.js is a small JavaScript library that helps you to communicate from your component's JavaScript with your JATOS server. Below we list and describe the variables and functions of the jatos.js library.

Have a look at what's mandatory in HTML and JavaScript for JATOS components (page 19). Always load the jatos.js script in the `<head>` section with the following line:

```
<script src="/assets/javascripts/jatos.js"></script>
```

All variables or calls to jatos.js start with `jatos.`. For example, if you want to get the study's ID you use `jatos.studyId`. If you submit result data of your component back to your JATOS server you use `jatos.submitResultData(resultData)`, where resultData can be any kind of text.

And, please, if you find a mistake or have a question don't hesitate to contact us (page 4).

jatos.js variables

You can call any of these variables below at any point in your HTML file after jatos.js finished initializing (`jatos.onload()` will be called). Most variables are read-only. A few variables can be written into (e.g. `jatos.httpTimeout`). Those are marked '(writeable)'.

IDs

All those IDs are generated and stored by JATOS. jatos.js automatically sets these variables with the corresponding values if you included the `jatos.onLoad()` callback function at the beginning of your JavaScript.

- `jatos.studyId` - ID of the study which is currently running. All the study properties are associated with this ID.
- `jatos.componentId` - ID of the component which is currently running. All the component properties are associated with this ID.
- `jatos.batchId` - ID of the batch this study run belongs to. All batch

properties are associated with this ID.

- `jatos.workerId` - Each worker who is running a study has an ID.
- `jatos.studyResultId` - This ID is individual for every study run. A study result contains data belonging to the run in general (e.g. Study Session).
- `jatos.componentResultId` - This ID is individual for every component in a study run. A component result contains data of the run belonging to the specific component (e.g. result data).
- `jatos.groupMemberId` - see Group Variables (page 103)
- `jatos.groupResultId` - see Group Variables (page 103)

There's a convenient function that adds all these IDs to a given object. See function `jatos.addJatosIds(obj)` below.

Study variables

- `jatos.studyProperties` - All the properties (except the JSON input data) you entered for this study
    - `jatos.studyProperties.title` - Study's title
    - `jatos.studyProperties.uuid` - Study's UUID
    - `jatos.studyProperties.description` - Study's description
    - `jatos.studyProperties.descriptionHash` - Hash of study's description
    - `jatos.studyProperties.locked` - Whether the study is locked or not
    - `jatos.studyProperties.dirName` - Study's dir name in the file system of your JATOS installation
    - `jatos.studyProperties.groupStudy` - Whether this is a group study or not
- `jatos.studyJsonInput` - The JSON input you entered in the study's properties.
- `jatos.studyLength` - Number of component this study has

Original URL query parameters

- `jatos.urlQueryParameters` - Original query string parameters of the URL that starts the study. It is provided as a JavaScript object. This might be useful to pass on information from outside of JATOS into a study run, e.g. if you want to pass on information like gender and age. However if you know the information beforehand it's easier to put them in the Study's or Component's JSON input. Another example is MTurk which passes on it's worker's ID via a URL query parameter.

Example: One has this link to start a Personal Single Run:

`http://localhost:9000/publix/50/start?batchId=47&personalSingleWorkerId=506`

Now one could add parameters to the URL's query string to pass on external information into the study run. E.g. the following URL would add the parameters 'foo' with the value 'bar' and 'a' with the value '123':

`http://localhost:9000/publix/50/start?batchId=47&personalSingleWorkerId=506&foo=bar&a=123`

Then those parameter will be accessible during the study run in `jatos.urlQueryParameters` as `{a: "123", foo: "bar"}`.

Example: MTurk uses for its worker ID the URL query parameter 'workerId' and this is accessible via `jatos.urlQueryParameters.workerId`.

Component variables

- `jatos.componentProperties` - All the properties (except the JSON input data) you entered for this component
    - `jatos.componentProperties.title` - Component's title
    - `jatos.componentProperties.uuid` - Component's UUID
    - `jatos.componentProperties.htmlFilePath` - Path to Component's HTML file in your JATOS installation
    - `jatos.componentProperties.reloadable` - Whether it's reloadable
- `jatos.componentJsonInput` - The JSON input you entered in the component's properties.
- `jatos.componentList` - An array of all components of this study with basic information about each component. For each component it has the `title`, `id`, whether it is `active`, and whether it is `reloadable`.
- `jatos.componentPos` - Position of this component within the study starting with 1 (like shown in the GUI)

Study’s session data

The session data can be accessed and modified by every component of a study. It’s a very convenient way to share data between different components. Whatever is written in this variable will be available in the subsequent components. However, remember that the session data will be deleted after the study is finished (see also Session Data - Three Types (page 28)).

- `jatos.studySessionData` (writeable)

Batch variables

- `jatos.batchProperties` - All the properties you entered for this batch.
    - `jatos.batchProperties.allowedWorkerTypes` - List of worker types that are currently allowed to run in this batch.
    - `jatos.batchProperties.maxActiveMembers` - How many members this group can have at the same time
    - `jatos.batchProperties.maxTotalMembers` - How many members this group is allowed to have at the same time
    - `jatos.batchProperties.maxTotalWorkers` - Total amount of workers this group is allowed to have altogether in this batch
    - `jatos.batchProperties.title` - Title of this batch
- `jatos.batchJsonInput` - The JSON input you entered in the batch’s properties.

Group variables

The group variables are part of jatos.js since JATOS 2. They are only filled with values if the current study is a group study.

- `jatos.groupMemberId` - Group member ID is unique for this member (it is actually identical with the study result ID)
- `jatos.groupResultId` - ID of this group result (It’s called group result to be consistent with the study result and the component result - although often it’s just called group)
- `jatos.groupState` - Represents the state of the group in JATOS; only set if group channel is open (one of STARTED, FIXED, FINISHED)
- `jatos.groupMembers` - List of member IDs of the current members of the group

- `jatos.groupChannels` - List of member IDs of the currently open group channels

Other variables

- `jatos.version` - Current version of the jatos.js library
- `jatos.channelSendingTimeoutTime` (writeable) - Time in ms to wait for an answer after sending a message via a channel (batch or group). Set this variable if you want to change the default value (default is 10 s).

```
jatos.channelSendingTimeoutTime = 20000; // Sets channe
l timeout to 20 seconds
```

- `jatos.httpTimeout` (writeable) - Time in ms to wait for an answer of an HTTP request by jatos.js. Set this variable if you want to change the default value (default is 1 min).

```
jatos.httpTimeout = 30000; // Sets HTTP timeout to 30 se
conds
```

- `jatos.httpRetry` - Some jatos functions (e.g. `jatos.sendResultData`) send an Ajax request to the JATOS server. If this request was not successful (e.g. network problems) jatos.js retries it. With this variable one can change the number of retries. The default is 5.

```
jatos.httpRetry = 2; // Attempts 2 retries of failed Aja
x requests
```

- `jatos.httpRetryWait` - Same as `jatos.httpRetry` but this variable defines the waiting time between the retries. The default is 1000 ms.

```
jatos.httpRetryWait = 5000; // Sets Ajax retry waiting t
ime to 5 seconds
```

- `jatos.jQuery` - You can always use jatos.js own jQuery if you want to save some bandwidth

General jatos.js functions

`jatos.onLoad(callback)`

Defines callback function that jatos.js will call when it's finished initialising. Only mandatory (page 19) call in every component.

- @param {Function} callback - function to be called after jatos.js' initialization is done

Example:

```
jatos.onLoad(function() {
  // Start here with your code that uses jatos.js' variables an
d functions
});
```

`jatos.onError(callback)`

Defines a callback function that is to be called in case jatos.js produces an error.

- @param {Function} callback - Function to be called in case of an error

Example that shows the error message in an alert box:

```
jatos.onError(function(error) {
  alert(error);
});
```

`jatos.log(logMsg)`

Sends a message to be logged back to the JATOS server where it will be logged in JATOS' log file.

- *@param {String} logMsg* - The messages to be logged

Example:

```
jatos.log("Log this message in JATOS' log file");
```

`jatos.addJatosIds(obj)`

Convenience function that adds some IDs (page 100) (study ID, study title, batch ID, batch title, component ID, component position, component title, worker ID, study result ID, component result ID, group result ID, group member ID) to the given object.

- *@param {Object} obj* - Object to which the IDs will be added

Example:

```
var resultData = {};
jatos.addJatosIds(resultData);
```

`jatos.setHeartbeatPeriod(heartbeatPeriod)`

Every running component sends regularly a HTTP request (the heartbeat) back to the JATOS server. This signals that it is still running. As soon as the browser tab running the component is closed the heartbeat ceases. The time of the last heartbeat is visible in the GUI, in the study results page in the 'Last Seen' row. This way you can easily see if a worker is still running your study or if (and when) he abandonend it. By default the heartbeat period is 2 minutes. By careful not to set the period too low (few seconds or even milliseconds) since it might overload your network or your JATOS server.

- *@param {Number} heartbeatPeriod* - Time period between two heartbeats in milliseconds

Example:

```
jatos.setHeartbeatPeriod(60000); // Sets to a heartbeat every m
inute
```

Functions to control study flow

`jatos.startComponent(componentId, resultData, onError)`

(Before v3.3.1 `jatos.startComponent(componentId)`)

Finishes the currently running component and starts the component with the given ID. Though often it's better to use `jatos.startComponentByPos` instead because it keeps working even after an export/import of the study. Since v3.3.1 one can additionally send result data back to the JATOS server.

- *@param {Number} componentId* - ID of the component to start
- *@param {optional Object} resultData* - String or Object that will be sent as result data. An Object will be serialized to JSON (stringify).
- *@param {optional Function} onError* - Callback function if fail

Examples:

```
jatos.startComponent(23); // Jumps to component with ID 23
```

It's often used together with `jatos.submitResultData` to first submit result data back to the JATOS server and afterwards jump to another component:

```
var resultData = "my important result data";
jatos.submitResultData(resultData, function() {
  jatos.startComponent(23);
});
```

Since v3.3.1 it's possible to use the shorter way to achieve the same:

```
var resultData = "my important result data";
jatos.startComponent(23, resultData);
```

`jatos.startComponentByPos(componentPos, resultData, onError)`

(Before v3.3.1 `jatos.startComponentByPos(componentPos)`)

Finishes the currently running component and starts the component with the given position. The component position is the count of the component within the study like shown in the study overview page (1st component has position 1, 2nd component position 2, …). Since v3.3.1 one can additionally send result data back to the JATOS server.

- *@param {Number} componentPos* - Position of the component to start
- *@param {optional Object} resultData* - String or Object that will be sent as result data. An Object will be serialized to JSON (stringify).

- *@param {optional Function} onError* - Callback function if fail

Examples:

```
jatos.startComponentByPos(3); // Jumps to component with positi
on 3
```

It's often used together with `jatos.submitResultData` to first submit result data back to the JATOS server and afterwards jump to another component:

```
var resultData = "my important result data";
jatos.submitResultData(resultData, function() {
  jatos.startComponentByPos(3);
});
```

Since v3.3.1 it's possible to use the shorter way to achieve the same:

```
var resultData = "my important result data";
jatos.startComponentByPos(3, resultData);
```

`jatos.startNextComponent(resultData, onError)`

(Before v3.3.1 `jatos.startNextComponent()`)

Finishes the currently running component and starts the next component of this study. The next component is the one with position + 1. The component position is the count of the component within the study like shown in the study overview page (1st component has position 1, 2nd component position 2, …). Since v3.3.1 one can additionally send result data back to the JATOS server.

- *@param {optional Object} resultData* - String or Object that will be sent as result data. An Object will be serialized to JSON (stringify).
- *@param {optional Function} onError* - Callback function if fail

Examples:

```
jatos.startNextComponent(); // Jumps to the next component
```

It's often used together with `jatos.submitResultData` to first submit result data back to the JATOS server and afterwards jump to the next component:

```
var resultData = "my important result data";
jatos.submitResultData(resultData, jatos.startNextComponent);
```

Since v3.3.1 it's possible to use the shorter way to achieve the same:

```
var resultData = "my important result data";
jatos.startNextComponent(resultData);
```

`jatos.startLastComponent(resultData, onError)`

(Before v3.3.1 `jatos.startLastComponent()`)

Finishes the current component and starts the last component of this study. If the last component is inactive it starts the component with the highest position that is active. The component position is the count of the component within the study like shown in the study overview page (1st component has position 1, 2nd component position 2, …). Since v3.3.1 one can additionally send result data back to the JATOS server.

- *@param {optional Object} resultData* - String or Object that will be sent as result data. An Object will be serialized to JSON (stringify).
- *@param {optional Function} onError* - Callback function if fail

Example:

```
jatos.startLastComponent(); // Jumps to the last component
```

It's often used together with `jatos.submitResultData` to first submit result data back to the JATOS server and afterwards jump to the next component:

```
var resultData = "my important result data";
jatos.submitResultData(resultData, jatos.startLastComponent);
```

Since v3.3.1 it's possible to use the shorter way to achieve the same:

```
var resultData = "my important result data";
jatos.startLastComponent(resultData);
```

`jatos.abortStudy(message)`

Aborts study. All previously submitted result data will be deleted. Afterwards the worker is redirected to the study end page. Data stored in the Batch Session or Group Session are uneffected by this.

- *@param {optional String} message* - Message that will be stored together with the study results and is accessible via JATOS' GUI result pages. The message can be max 255 characters long.

Example:

```
jatos.abortStudy();
```

Or:

```
jatos.abortStudy("participant aborted by pressing abort butto
n");
```

`jatos.endStudyAjax(successful, message, onSuccess, onError)`

Ends study with an Ajax call - afterwards the study is not redirected to the JATOS' end page. It offers callbacks, either as parameter or via jQuery.deferred.promise, to signal success or failure in the ending.

- *@param {optional Boolean} successful* - 'true' if study should finish successful and the participant should get the confirmation code - 'false' otherwise.
- _@param {optional String} message - Message that will be stored together with the study results and is accessible via JATOS' GUI result pages. The message can be max 255 characters long.
- *@param {optional Function} onSuccess* - Function to be called in case of successful submit
- *@param {optional Function} onError* - Function to be called in case of error
- *@return {jQuery.deferred.promise}*

Example:

```
jatos.endStudyAjax();
```

Or:

```
jatos.endStudyAjax("everything worked fine");
```

Or to indicate a failure:

```
jatos.endStudyAjax(false, "internal JS error");
```

`jatos.endStudy(successful, message)`

Ends study. Redirects the worker to the study end page afterwards.

- *@param {optional Boolean} successful* - 'true' if study should finish successfully, 'false' otherwise. Default is true.
- *@param {optional String} msg* - Message that will be stored together with the study results and is accessible via JATOS' GUI result pages. The message can be max 255 characters long.

Example:

```
jatos.endStudy();
```

Or:

```
jatos.endStudy("everything worked fine");
```

Or to indicate a failure:

```
jatos.endStudyAjax(false, "internal JS error");
```

Functions for Study Session and result data

`jatos.submitResultData(resultData, onSuccess, onError)`

Posts result data for the currently running component back to the JATOS server. Already stored result data for this component will be **overwritten**. If you want to append result data use `jatos.appendResultData` instead. It offers callbacks, either as parameter or via jQuery.deferred.promise, to signal success or failure in the transfer.

- *@param {Object} resultData* - String or Object that will be sent as result data. An Object will be serialized to JSON.
- *@param {optional Function} success* - Function to be called in case of successful submit
- *@param {optional Function} error* - Function to be called in case of error
- *@return {jQuery.deferred.promise}*

Examples:

```
var resultData = {"a": 123, "b": 789, "c": 100};
jatos.submitResultData(JSON.stringify(resultData));
```

It's often used together with `jatos.startNextComponent` to first submit result data back to the JATOS server and afterwards jump to the next component:

```
var resultData = {"a": 123, "b": 789, "c": 100};
jatos.submitResultData(JSON.stringify(resultData), jatos.startN
extComponent);
```

Or together with `jatos.startComponentByPos` to start a particular component (here at position 4):

```
var resultData = {"a": 123, "b": 789, "c": 100};
jatos.submitResultData(JSON.stringify(resultData), function ()
{
  jatos.startComponentByPos(4);
});
```

Since v3.3.1 it’s possible to leave out the JSON serialization:

```
var resultData = {"a": 123, "b": 789, "c": 100};
jatos.submitResultData(resultData, jatos.startNextComponent);
```

`jatos.appendResultData(resultData, onSuccess, onError)`

Since JATOS version >= 3.1.7

Appends result data to the already posted result data. Contrary to jatos.submitResultData it does not overwrite the result data. It offers callbacks, either as parameter or via jQuery.deferred.promise, to signal success or failure in the transfer. This function can be used several times during an component run to incrementally save result data.

- *@param {String} resultData* - String or Object that will be sent as result data. An Object will be serialized to JSON (stringify).
- *@param {optional Function} success* - Function to be called in case of successful submit
- *@param {optional Function} error* - Function to be called in case of error
- *@return {jQuery.deferred.promise}*

Examples:

```
var resultData = { "a": 123, "b": 789, "c": 100};
jatos.appendResultData(JSON.stringify(resultData));
```

You can use it together with `jatos.startNextComponent` to first append result data and afterwards jump to the next component:

```
var resultData = { "a": 123, "b": 789, "c": 100};
jatos.appendResultData(JSON.stringify(resultData), jatos.startN
extComponent);
```

Or together with `jatos.startComponentByPos` to start a particular component (here at position 4):

```
var resultData = { "a": 123, "b": 789, "c": 100};
jatos.appendResultData(JSON.stringify(resultData), function ()
{
  jatos.startComponentByPos(4);
});
```

Since v3.3.1 it’s possible to use the shorter way to achieve the same:

```
var resultData = {"a": 123, "b": 789, "c": 100};
jatos.startNextComponent(resultData);
```

Or:

```
var resultData = {"a": 123, "b": 789, "c": 100};
jatos.startComponentByPos(3, resultData);
```

`jatos.setStudySessionData(studySessionData, onSuccess, onFail)`

Posts Study Session data to the JATOS server. This function is called automatically in the end of a component’s life cycle (it’s called by all jatos.js functions that end a component). So unless you want to store the session data in between a component run it’s not necessary to call this function manually. It offers callbacks, either as parameter or via jQuery.deferred.promise, to signal success or failure in the transfer.

- *@param {Object} sessionData* - Object to be submitted
- *@param {optional Function} onSuccess* - Function to be called after this function is finished
- *@param {optional Function} onFail* - Function to be called after if this this functions fails
- *@return {jQuery.deferred.promise}*

Example:

```
var studySessionData = { "a": 123, "b": 789, "c": 100};
jatos.setStudySessionData(studySessionData);
```

Functions to access the Batch Session

The Batch Session is stored in JATOS' database on the server side (see also Session Data - Three Types (page 28)). That means that all changes in the Batch Session have to be synchronized between the client and the server. This is done via the batch channel. Therefore all writing functions (`add`, `remove`, `clear`, `replace`, `copy`, `move`, `set`, `setAll`) can be paired with callback functions that will signal success or failure in the client-server sync. These callback functions can be either passed as parameters to `jatos.batchSession.[function_name]` or via jQuery.deferred.promise.

On the other side for all reading functions (`get`, `find`, `getAll`, `test`) there is no need to sync data between client and server, because jatos.js keeps a copy of the Batch Session locally. Therefore all reading functions do not offer callbacks, because there is no risk of failure of synchronization.

Additionally to the reading and writing functions the calback function `jatos.onBatchSession(callback)` offers a way to get notified whenever the Batch Session changes in the JATOS' database regardless of the origin of the change. This way, you can have the client of each worker react to changes in the batch that were done by another worker in the batch.

Accessing the Batch Session is done via JSON Patches (RFC 6902) and JSON Pointer (RFC 6901). An introduction can be found under jsonpatch.com. For JSON Patches jatos.js uses the JSON-Patch library from Joachim Wester and for JSON Pointers the jsonpointer.js library from Alexey Kuzmin.

`jatos.batchSession.add(path, value, onSuccess, onFail)`

JSON Patch add operation: Adds a value to an object or inserts it into an array. In the case of an array, the value is inserted before the given index. The `-` character can be used instead of an index to insert at the end of an array (see jsonpatch.com). If the path already exists in the Batch Session the value will be overwritten.

- *@param {string} path* - JSON pointer path
- *@param {object} value* - value to be stored
- *@param {optional callback} onSuccess* - Function to be called if this patch was successfully applied on the server and the client side
- *@param {optional callback} onError* - Function to be called if this patch failed

- *@return {jQuery.deferred.promise}*

Example: If the internal Batch Session is `{"a": 100}` and one calls

```
jatos.batchSession.add("/b", 123);
```

then after the Batch Session is successfully updated the new internal object is `{"a": 100, "b": 123}`.

Since there is a slight chance that the session update was not successful it’s a good idea to provide callback functions for both cases. To provide success or fail callback functions you can either specify the onSuccess/onError parameters or use jQuery’ deferred object.

Example with jQuery’s defered:

```
var deferred = jatos.batchSession.add("/b", 123);
deferred.done(function () {
  alert("Batch Session was successfully updated");
});
deferred.fail(function () {
  alert("Batch Session synchronization failed");
});
```

`jatos.batchSession.remove(path, onSuccess, onFail)`

JSON Patch remove operation: Removes a value from an object or array (see jsonpatch.com).

- *@param {string} path* - JSON pointer path to the field that should be removed
- *@param {optional callback} onSuccess* - Function to be called if this patch was successfully applied on the server and the client side
- *@param {optional callback} onError* - Function to be called if this patch failed
- *@return {jQuery.deferred.promise}*

Example: If the internal Batch Session is `{"a": 100, "b": 123}` and one calls

```
jatos.batchSession.remove("/b");
```

then after the Batch Session is successfully updated the new internal object is `{"a": 100}`.

Since there is a slight chance that the session update was not successful it's a good idea to provide callback functions for both cases. To provide success or fail callback functions you can either specify the onSuccess/onError parameters or use jQuery' deferred object.

Example with jQuery's defered:

```
var deferred = jatos.batchSession.remove("/b");
deferred.done(function () {
  alert("Batch Session was successfully updated");
});
deferred.fail(function () {
  alert("Batch Session synchronization failed");
});
```

`jatos.batchSession.replace(path, value, onSuccess, onFail)`

JSON Patch replace operation: Replaces a value. Equivalent to a 'remove' followed by an 'add' (see jsonpatch.com).

- *@param {string} path* - JSON pointer path
- *@param {object} value* - value to be replaced with
- *@param {optional callback} onSuccess* - Function to be called if this patch was successfully applied on the server and the client side
- *@param {optional callback} onError* - Function to be called if this patch failed
- *@return {jQuery.deferred.promise}*

Example: If the internal Batch Session is `{"a": 100, "b": 123}` and one calls

```
jatos.batchSession.replace("/b", 789);
```

then after the Batch Session is successfully updated the new internal object is `{"a": 100, "b": 789}`.

Since there is a slight chance that the session update was not successful it's a good idea to provide callback functions for both cases. To provide success or fail callback functions you can either specify the onSuccess/onError parameters or use jQuery' deferred object.

Example with jQuery's defered:

```
var deferred = jatos.batchSession.replace("/b", 789);
deferred.done(function () {
  alert("Batch Session was successfully updated");
});
deferred.fail(function () {
  alert("Batch Session synchronization failed");
});
```

`jatos.batchSession.copy(from, path, onSuccess, onFail)`

JSON Patch copy operation: Copies a value from one location to another within the JSON document. Both from and path are JSON Pointers (see jsonpatch.com).

- *@param {string} from* - JSON pointer path to the origin
- *@param {string} path* - JSON pointer path to the target
- *@param {optional callback} onSuccess* - Function to be called if this patch was successfully applied on the server and the client side
- *@param {optional callback} onError* - Function to be called if this patch failed
- *@return {jQuery.deferred.promise}*

Example: If the internal Batch Session is `{"a": "jatos"}` and one calls

```
jatos.batchSession.copy("/a", "/b");
```

then after the Batch Session is successfully updated the new internal object is `{"a": "jatos", "b": "jatos"}`.

Since there is a slight chance that the session update was not successful it's a good idea to provide callback functions for both cases. To provide success or fail callback functions you can either specify the onSuccess/onError parameters or use jQuery' deferred object.

Example with jQuery's defered:

```
var deferred = jatos.batchSession.copy("/a", "/b");
deferred.done(function () {
  alert("Batch Session was successfully updated");
});
deferred.fail(function () {
  alert("Batch Session synchronization failed");
});
```

`jatos.batchSession.move(from, path, onSuccess, onFail)`

JSON Patch move operation: Moves a value from one location to the other. Both from and path are JSON Pointers. (see jsonpatch.com).

- *@param {string} from* - JSON pointer path to the origin
- *@param {string} path* - JSON pointer path to the target
- *@param {optional callback} onSuccess* - Function to be called if this patch was successfully applied on the server and the client side
- *@param {optional callback} onError* - Function to be called if this patch failed
- *@return {jQuery.deferred.promise}*

Example: If the internal Batch Session is `{"a": "jatos"}` and one calls

```
jatos.batchSession.move("/a", "/b");
```

then after the Batch Session is successfully updated the new internal object is `{"b": "jatos"}`.

Since there is a slight chance that the session update was not successful it's a good idea to provide callback functions for both cases. To provide success or fail callback functions you can either specify the onSuccess/onError parameters or use jQuery' deferred object.

Example with jQuery's defered:

```
var deferred = jatos.batchSession.move("/a", "/b");
deferred.done(function () {
  alert("Batch Session was successfully updated");
});
deferred.fail(function () {
  alert("Batch Session synchronization failed");
});
```

`jatos.batchSession.find(path)`

Gets a field in the Batch Session data. Takes a JSON Pointer and returns the matching value. Gets the object from the locally stored copy of the session and does not call the server. Contrary to `jatos.batchSession.get` it allows to get values from all levels of the Batch Session's object tree.

- *@param {string} path* - JSON pointer path
- *@return {object}* - the value that is stored in path

Example: If the internal Batch Session is `{"a": {"a1": "foo", "a2": "bar"}, "b": 999}`

```
jatos.batchSession.find("/a/a1"); // returns "foo"
jatos.batchSession.find("/b"); // returns 999
```

`jatos.batchSession.defined(path)`

Since JATOS version >= 3.1.8

Checks in the Batch Session whether a field under the given path exists. Returns true if the field is defined and false otherwise. It's equivalent to `!jatos.batchSession.test(path, undefined)`.

- *@param {string} path* - JSON pointer path to be checked
- *@return {boolean}*

Example:

```
jatos.batchSession.defined("/a"); // returns true if the pointe
r '/a' exists
```

`jatos.batchSession.test(path, value)`

JSON Patch test operation: Tests that the specified value is set in the document (see jsonpatch.com).

- *@param {string} path* - JSON pointer path to be tested
- *@param {object} value* - value to be tested
- *@return {boolean}*

Example: If the internal Batch Session is `{"a": 123, "b": {"b1": "flowers", "b2": "animals"}}`

```
jatos.batchSession.test("/a", 123); // returns true
jatos.batchSession.test("/a", 10); // returns false
jatos.batchSession.test("/b/b1", "flowers"); // returns true
```

Example: If you want to know the existence of a path in the Batch Session you can test against `undefined`:

```
if (!jatos.batchSession.test("/c", undefined)) {
  // Path "/c" exists
} else {
  // Path "/c" doesn't exist
}
```

`jatos.batchSession.get(name)`

Convenience function: like `jatos.batchSession.find` but works with a key instead of a JSON Pointer. Therefore it works only on the first level of the session's object tree. It takes a name of an field within the Batch Session and returns the matching value. For all other levels of the object tree use jatos.batchSession.find. Gets the object from the locally stored copy of the session and does not call the server.

- *@param {string} name* - name of the field
- *@return {object}* - the value that is stored under name

Example: If the internal Batch Session is `{"a": 1000, "b": "watermelon"}`

```
// Since the parameter is the key's name and not a path it doe
s not start with a "/"
var b = jatos.batchSession.get("b"); // b is "watermelon"
var c = jatos.batchSession.get("c"); // c is undefined
```

Example: With `jatos.batchSession.get` you can only access the first level of the object tree - if you want another level use `jatos.batchSession.find`. If the internal Batch Session is `{"a": {"a1": 123, "a2": "watermelon"}}`

```
var a1 = jatos.batchSession.get("a1"); // a1 is undefined !!!
var a = jatos.batchSession.get("a"); // a is { "a1": 123, "a
2": "watermelon" }
```

`jatos.batchSession.set(name, value, onSuccess, onFail)`

Like `jatos.batchSession.add`, but instead of a JSON Pointer path it accepts a name of the field to be stored. Therefore it works only on the first level of the Batch Session's object tree. If the name already exists in the Batch Session the value will be overwritten.

- *@param {string} name* - name of the field
- *@param {object} value* - value to be stored
- *@param {optional callback} onSuccess* - Function to be called if this patch was successfully applied on the server and the client side
- *@param {optional callback} onError* - Function to be called if this patch failed
- *@return {jQuery.deferred.promise}*

Example: If the internal Batch Session is `{"a": 1234}`

```
// Since the parameter is the key's name and not a path it doe
s not start with a "/"
var b = jatos.batchSession.set("b", "koala");
```

then after the Batch Session is successfully updated the new internal object is `{"a": 1234, "b": "koala"}`.

Since there is a slight chance that the session update was not successful it’s a good idea to provide callback functions for both cases. To provide success or fail callback functions you can either specify the onSuccess/onError parameters or use jQuery’ deferred object.

Example with jQuery’s defered:

```
var deferred = jatos.batchSession.set("b", "koala");
deferred.done(function () {
  alert("Batch Session was successfully updated");
});
deferred.fail(function () {
  alert("Batch Session synchronization failed");
});
```

`jatos.batchSession.getAll()`

Returns the complete Batch Session data. Gets the object from the locally stored copy of the session and does not call the server.

- *@return {object}*

Example:

```
var batchSession = jatos.batchSession.getAll();
```

`jatos.batchSession.setAll(value, onSuccess, onFail)`

Replaces the whole session data. If the replacing object is rather large it might be better performance-wise to replace only individual paths. Each session writting involves sending the changes in the session via a JSON Patch to the JATOS server. If the session is large this data transfer can take some time. In this case use other session functions, like ‘set’, ‘add’, or ‘replace’.

- *@param {object} value* - value to be stored in the session
- *@param {optional callback} onSuccess* - Function to be called if this patch was successfully applied on the server and the client side
- *@param {optional callback} onError* - Function to be called if this patch failed
- *@return {jQuery.deferred.promise}*

Example:

```
var o = {"a": 123, "b": "foo"};
jatos.batchSession.setAll(o); // Overwrites the current Batch S
ession with the object o
```

Since there is a slight chance that the session update was not successful it's a good idea to provide callback functions for both cases. To provide success or fail callback functions you can either specify the onSuccess/onError parameters or use jQuery' deferred object.

Example with jQuery's defered:

```
var o = {"a": 123, "b": "foo"};
var deferred = jatos.batchSession.setAll(o);
deferred.done(function () {
  alert("Batch Session was successfully updated");
});
deferred.fail(function () {
  alert("Batch Session synchronization failed");
});
```

`jatos.batchSession.clear(onSuccess, onFail)`

Clears the whole Batch Session data and sets it to an empty object `{}`.

- *@param {optional callback} onSuccess* - Function to be called if this patch was successfully applied on the server and the client side
- *@param {optional callback} onError* - Function to be called if this patch failed
- *@return {jQuery.deferred.promise}*

Example:

```
jatos.batchSession.clear();
```

Since there is a slight chance that the session update was not successful it's a good idea to provide callback functions for both cases. To provide success or fail callback functions you can either specify the onSuccess/onError parameters or use jQuery' deferred object.

Example with jQuery's defered:

```
var deferred = jatos.batchSession.clear();
deferred.done(function () {
  alert("Batch Session was successfully updated");
});
deferred.fail(function () {
  alert("Batch Session synchronization failed");
});
```

`jatos.onBatchSession(callback)`

Defines a callback function that is called every time the Batch Session changes on the JATOS server side (that includes updates in the session originating from other workers that run the study in parallel).

The callback function has two parameter (before v3.3.1 one parameter):

- *@param {String} path* - JSON pointer to the changed field in the Batch Session
- *@param {String} op* - (version >= 3.3.1) JSON patch operation ('add', 'remove', 'clear', …) that was applied

Example:

```
jatos.onBatchSession(function(path, op){
  alert("Batch Session was updated in path " + path + " with op
eration " + op);
});
```

Example: `onBatchSession` is often used together with `jatos.batchSession.find` to get the updated value:

```
jatos.onBatchSession(function(path){
  var changedObj = jatos.batchSession.find(path);
  alert("The changed object is " + JSON.stringify(changedObj));
});
```

Functions for group studies

`jatos.joinGroup(callbacks)`

Tries to join a group and if it succeeds opens the group channel (which is mostly a WebSocket). Only if the group channel is open one can exchange data with other group members. As the only parameter this function takes an object that consists of several optional callback functions that will be called by jatos.js when certain group events occur. It returns a jQuery.deferred.promise, to signal success or failure in joining.

- *@param {Object} callbacks* - Defining callback functions for group events. All callbacks are optional. These callbacks functions are:
    - `onOpen`: Is called when the group channel is successfully opened
    - `onClose`: Is be called when the group channel is closed
    - `onError`: Is called if an error during opening of the group channel's WebSocket occurs or if an error is received via the group channel (e.g. the Group Session data couldn't be updated). If this function is not defined jatos.js will try to call the global `onJatosError` function.
    - `onMessage(msg)`: Is called if a message from another group member is received. It gets the message as a parameter.
    - `onMemberJoin(memberId)`: Is called when another member (not the worker running this study) joined the group. It gets the group member ID as a parameter.
    - `onMemberOpen(memberId)`: Is called when another member (not the worker running this study) opened a group channel. It gets the group member ID as a parameter.
    - `onMemberLeave(memberId)`: Is called when another member (not the worker running his study) left the group. It gets the group member ID as a parameter.
    - `onMemberClose(memberId)`: Is called when another member (not the worker running this study) closed his group channel. It gets the group member ID as a parameter.
    - `onGroupSession(path, op)`: Is called every time the Group Session changes on the JATOS server side. It gets two

parameters (before v3.3.1 only one): 1) JSON pointer path to the changed field in the Group Session as a parameter, and 2) JSON patch operation.

    - `onUpdate()`: Combines several other callbacks. It's called if one of the following is called: `onMemberJoin`, `onMemberOpen`, `onMemberLeave`, `onMemberClose`, or `onGroupSession`.

- *@return {jQuery.deferred.promise}*

Minimal example that joins a group and receives updates via the Group Session:

```
jatos.joinGroup({
  "onGroupSession": onGroupSession
});

function onGroupSession(path, op) {
   var changedObj = jatos.groupSession.find(path);
   alert("Group Session was updated in path " + path + " with o
peration " + op + " to " + JSON.stringify(changedObj));
}
```

Example that defines the `onOpen`, `onMemberOpen`, and `onMessage` callbacks:

```
jatos.joinGroup({
  "onOpen": onOpen,
  "onMemberOpen": onMemberOpen,
  "onMessage": onMessage
});

function onOpen() {
  alert("You joined a group and opened a group channel");
}

function onMemberOpen(memberId) {
  alert("In our group another member (ID " + memberId + ") open
ed a group channel");
}

function onMessage(msg) {
   alert("You received a message: " + msg);
}
```

`jatos.sendGroupMsg(msg)`

Sends a message to all group members with an open group channel. Use `jatos.sendGroupMsgTo` to send a message to a particular member.

Between group members data can be exchanged in fundamentally two different ways: sendGroupMsg/sendGroupMsgTo or the Group Session (page 132). The main difference is that the Group Session is stored in JATOS database on the server side while with sendGroupMsg/sendGroupMsgTo the data are only relayed on the server side but is never stored. E.g. if the worker reloads the page all prior messages sent by sendGroupMsg/sendGroupMsgTo will be lost - on the other side, everything stored in the Group Session will be restored. But this storage of the Group Session in JATOS comes at the cost of being (slightly) slower. Which option to choose depends mostly on your study design. If you expect your workers to have an unreliable Internet connection or to reload the page then you should use the Group Session. If you just want to 'stream' current data to other members the use sendGroupMsg/sendGroupMsgTo.

- *@param {Object} msg* - Any JavaScript object

Examples:

```
var msg = "Message for every group member"; // Send a text mess
age
jatos.sendGroupMsg(msg)

var objMsg = {"city": "Berlin", "population": 3500000"}; // Sen
d an object
jatos.sendGroupMsg(objMsg)
```

`jatos.sendGroupMsgTo(recipient, msg)`

Like `jatos.sendGroupMsg` but sends a message to a particular group member specified by the group member ID. You can find a list of all IDs of group members with an open channel `jatos.groupChannels`. Alternativally you get member IDs via the `onMemberOpen` callback function.

- *@param {String} recipient* - Recipient's group member ID
- *@param {Object} msg* - Any JavaScript object

Example:

```
var msg = "Message for group member 1063";
jatos.sendGroupMsgTo("1063", msg)
```

In the next example we use the `onMemberOpen` callback to send a message right after a new member opened their group channel:

```
jatos.joinGroup({
  "onMemberOpen": onMemberOpen,
  "onMessage": onMessage
});

function onMemberOpen(memberId) {
  var msg = "Welcome to the group!";
  jatos.sendGroupMsgTo(memberId, msg);
}

function onMessage(msg) {
   alert("You received a message: " + msg);
}
```

`jatos.leaveGroup(onSuccess, onError)`

Leaves the group it has previously joined. It offers callbacks, either as parameter or via jQuery.deferred.promise, to signal success or failure in the leaving.

- *@param {optional Function} onSuccess* - Function to be called after the group is left
- *@param {optional Function} onError* - Function to be called in case of error
- *@return {jQuery.deferred.promise}*

Example:

```
jatos.leaveGroup();
```

`jatos.reassignGroup(onSuccess, onFail)`

Asks the JATOS server to reassign this study run to a different group. JATOS can only reassign if there is another group availible. It offers callbacks, either as parameter or via jQuery.deferred.promise, to signal success or failure in the reassigning.

- *@param {optional Function} onSuccess* - Function to be called if the reassignment was successful
- *@param {optional Function} onFail* - Function to be called if the reassignment was unsuccessful
- *@return {jQuery.deferred.promise}*

Example:

```
var deferred = jatos.reassignGroup();
deferred.done(function () {
  alert("Successful group reassignment: new group ID is " + jat
os.groupResultId);
});
deferred.fail(function () {
  alert("Group reassignment failed");
});
```

`jatos.setGroupFixed()`

Ask the JATOS server to fix this group. A fixed group is not allowed to take on more members although members are still allowed to leave. It offers callbacks, either as parameter or via jQuery.deferred.promise, to signal success or failure in the fixing.

- *@param {optional Function} onSuccess* - Function to be called if the fixing was successful
- *@param {optional Function} onFail* - Function to be called if the fixing was unsuccessful
- *@return {jQuery.deferred.promise}*

Example:

```
jatos.setGroupFixed();
```

`jatos.hasJoinedGroup()`

Returns true if this study run joined a group and false otherwise. It doesn't necessarily mean that we have an open group channel. We might just have joined a group in a prior component but in this component never opened the channel. If you want to check for an open group channel use `jatos.hasOpenGroupChannel`.

Example:

```
if(jatos.hasJoinedGroup()) {
  // We are member in a group
} else {
  // We are not member in a group
};
```

`jatos.hasOpenGroupChannel()`

Returns true if we currently have an open group channel and false otherwise. Since you can't open a group channel without joining a group, it also means that we joined a group. On the other side although we have closed group channel we can still be a member in a group. Use `jatos.hasJoinedGroup` to check group membership.

Example:

```
if(jatos.hasOpenGroupChannel()) {
  // We are member in a group and have an open group channel
} else {
  // We do not have an open group channel (but could still be m
ember in a group)
};
```

`jatos.isMaxActiveMemberReached()`

Returns true if the group has reached the maximum amount of active members like specified in the batch properties. It's not necessary that each member has an open group channel.

Example:

```
if(jatos.isMaxActiveMemberReached()) {
  // Maximum number of active members is reached
};
```

`jatos.isMaxActiveMemberOpen()`

Returns true if the group has reached the maximum amount of active members like specified in the batch properties and each member has an open group channel.

```
if(jatos.isMaxActiveMemberOpen()) {
  // Maximum number of active members is reached and each has a
n open channel
};
```

`jatos.isGroupOpen()`

Returns true if all active members of the group have an open group channel and can send and receive data. It’s not necessary that the group has reached its minimum or maximum active member size.

```
if(jatos.isGroupOpen()) {
  // Each of the current members of the group have an open grou
p channel
};
```

Functions to access the Group Session

The Group Session is one of three way to communicate between members of a group. The others are direct messaging (with jatos.sendGroupMsgTo (page 128)) and broadcast messaging (jatos.sendGroupMsg (page 128)) (or: more general information about the different session types (page 0)).

In difference to the Batch Session (page 115) the Group Session doesn’t work from the start of a component. To use the Group Session you have to join a group (with jatos.joinGroup (page 126)). There you can also define a `onGroupSession` callback that gets called each time the Group Session changes regardless of the origin of the change.

The Group Session is stored in JATOS' database on the server side. That means that all changes in the Group Session have to be synchronized between the client and the server. This is done via the group channel. Therefore all writing functions (`add`, `remove`, `clear`, `replace`, `copy`, `move`, `set`, `setAll`) can be paired with callback functions that will signal success or failure in the client-server sync. These callback functions can be either passed as parameters to `jatos.groupSession.[function_name]` or via jQuery.deferred.promise.

On the other side for all reading functions (`get`, `find`, `getAll`, `test`) there is no need to sync data between client and server, because jatos.js keeps a copy of the Group Session locally. Therefore all reading functions do not offer callbacks, because there is no risk of failure of synchronization.

Accessing the Group Session is done via JSON Patches (RFC 6902) and JSON Pointer (RFC 6901). An introduction can be found under jsonpatch.com. For JSON Patches jatos.js uses the JSON-Patch library from Joachim Wester and for JSON Pointers the jsonpointer.js library from Alexey Kuzmin.

`jatos.groupSession.add(path, value, onSuccess, onFail)`

JSON Patch add operation: Adds a value to an object or inserts it into an array. In the case of an array, the value is inserted before the given index. The `-` character can be used instead of an index to insert at the end of an array (see jsonpatch.com). If the path already exists in the Group Session the value will be overwritten.

- *@param {string} path* - JSON pointer path
- *@param {object} value* - value to be stored
- *@param {optional callback} onSuccess* - Function to be called if this patch was successfully applied on the server and the client side
- *@param {optional callback} onError* - Function to be called if this patch failed
- *@return {jQuery.deferred.promise}*

Example: If the internal Group Session is `{"a": 100}` and one calls

```
jatos.groupSession.add("/b", 123);
```

then after the Group Session is successfully updated the new internal object is `{"a": 100, "b": 123}`.

Since there is a slight chance that the session update was not successful it's a good idea to provide callback functions for both cases. To provide success or fail callback functions you can either specify the onSuccess/onError parameters or use jQuery' deferred object.

Example with jQuery's defered:

```
var deferred = jatos.groupSession.add("/b", 123);
deferred.done(function () {
  alert("Group Session was successfully updated");
});
deferred.fail(function () {
  alert("Group Session synchronization failed");
});
```

`jatos.groupSession.remove(path, onSuccess, onFail)`

JSON Patch remove operation: Removes a value from an object or array (see jsonpatch.com).

- *@param {string} path* - JSON pointer path to the field that should be removed
- *@param {optional callback} onSuccess* - Function to be called if this patch was successfully applied on the server and the client side
- *@param {optional callback} onError* - Function to be called if this patch failed
- *@return {jQuery.deferred.promise}*

Example: If the internal Group Session is `{"a": 100, "b": 123}` and one calls

```
jatos.groupSession.remove("/b");
```

then after the Group Session is successfully updated the new internal object is `{"a": 100}`.

Since there is a slight chance that the session update was not successful it's a good idea to provide callback functions for both cases. To provide success or fail callback functions you can either specify the onSuccess/onError parameters or use jQuery' deferred object.

Example with jQuery's defered:

```
var deferred = jatos.groupSession.remove("/b");
deferred.done(function () {
  alert("Group Session was successfully updated");
});
deferred.fail(function () {
  alert("Group Session synchronization failed");
});
```

`jatos.groupSession.replace(path, value, onSuccess, onFail)`

JSON Patch replace operation: Replaces a value. Equivalent to a “remove” followed by an “add” (see jsonpatch.com).

- *@param {string} path* - JSON pointer path
- *@param {object} value* - value to be replaced with
- *@param {optional callback} onSuccess* - Function to be called if this patch was successfully applied on the server and the client side
- *@param {optional callback} onError* - Function to be called if this patch failed
- *@return {jQuery.deferred.promise}*

Example: If the internal Group Session is `{"a": 100, "b": 123}` and one calls

```
jatos.groupSession.replace("/b", 789);
```

then after the Group Session is successfully updated the new internal object is `{"a": 100, "b": 789}`.

Since there is a slight chance that the session update was not successful it’s a good idea to provide callback functions for both cases. To provide success or fail callback functions you can either specify the onSuccess/onError parameters or use jQuery’ deferred object.

Example with jQuery’s defered:

```
var deferred = jatos.groupSession.replace("/b", 789);
deferred.done(function () {
  alert("Group Session was successfully updated");
});
deferred.fail(function () {
  alert("Group Session synchronization failed");
});
```

`jatos.groupSession.copy(from, path, onSuccess, onFail)`

JSON Patch copy operation: Copies a value from one location to another within the JSON document. Both from and path are JSON Pointers (see jsonpatch.com).

- *@param {string} from* - JSON pointer path to the origin
- *@param {string} path* - JSON pointer path to the target
- *@param {optional callback} onSuccess* - Function to be called if this patch was successfully applied on the server and the client side
- *@param {optional callback} onError* - Function to be called if this patch failed
- *@return {jQuery.deferred.promise}*

Example: If the internal Group Session is `{"a": "jatos"}` and one calls

```
jatos.groupSession.copy("/a", "/b");
```

then after the Group Session is successfully updated the new internal object is `{"a": "jatos", "b": "jatos"}`.

Since there is a slight chance that the session update was not successful it's a good idea to provide callback functions for both cases. To provide success or fail callback functions you can either specify the onSuccess/onError parameters or use jQuery' deferred object.

Example with jQuery's defered:

```
var deferred = jatos.groupSession.copy("/a", "/b");
deferred.done(function () {
  alert("Group Session was successfully updated");
});
deferred.fail(function () {
  alert("Group Session synchronization failed");
});
```

`jatos.groupSession.move(from, path, onSuccess, onFail)`

JSON Patch move operation: Moves a value from one location to the other. Both from and path are JSON Pointers. (see jsonpatch.com).

- *@param {string} from* - JSON pointer path to the origin
- *@param {string} path* - JSON pointer path to the target
- *@param {optional callback} onSuccess* - Function to be called if this patch was successfully applied on the server and the client side
- *@param {optional callback} onError* - Function to be called if this patch failed
- *@return {jQuery.deferred.promise}*

Example: If the internal Group Session is `{"a": "jatos"}` and one calls

```
jatos.groupSession.move("/a", "/b");
```

then after the Group Session is successfully updated the new internal object is `{"b": "jatos"}`.

Since there is a slight chance that the session update was not successful it's a good idea to provide callback functions for both cases. To provide success or fail callback functions you can either specify the onSuccess/onError parameters or use jQuery' deferred object.

Example with jQuery's defered:

```
var deferred = jatos.groupSession.move("/a", "/b");
deferred.done(function () {
  alert("Group Session was successfully updated");
});
deferred.fail(function () {
  alert("Group Session synchronization failed");
});
```

`jatos.groupSession.find(path)`

Gets a field in the Group Session data. Takes a JSON Pointer and returns the matching value. Gets the object from the locally stored copy of the session and does not call the server. Contrary to `jatos.groupSession.get` it allows to get values from all levels of the Group Session's object tree.

- *@param {string} path* - JSON pointer path
- *@return {object}* - the value that is stored in path

Example: If the internal Group Session is `{"a": {"a1": "foo", "a2": "bar"}, "b": 999}`

```
jatos.groupSession.find("/a/a1"); // returns "foo"
jatos.groupSession.find("/b"); // returns 999
```

`jatos.groupSession.defined(path)`

Since JATOS version >= 3.1.8

Checks in the Group Session whether a field under the given path exists. Returns true if the field is defined and false otherwise. It's equivalent to `!jatos.groupSession.test(path, undefined)`.

- *@param {string} path* - JSON pointer path to be checked
- *@return {boolean}*

Example:

```
jatos.groupSession.defined("/a"); // returns true if the pointe
r '/a' exists
```

`jatos.groupSession.test(path, value)`

JSON Patch test operation: Tests that the specified value is set in the document (see jsonpatch.com).

- *@param {string} path* - JSON pointer path to be tested
- *@param {object} value* - value to be tested
- *@return {boolean}*

Example: If the internal Group Session is `{"a": 123, "b": {"b1": "flowers", "b2": "animals"}}`

```
jatos.groupSession.test("/a", 123); // returns true
jatos.groupSession.test("/a", 10); // returns false
jatos.groupSession.test("/b/b1", "flowers"); // returns true
```

`jatos.groupSession.get(name)`

Convenience function: like `jatos.groupSession.find` but works with a key instead of a JSON Pointer. Therefore it works only on the first level of the session's object tree. It takes a name of an field within the Group Session and returns the matching value. For all other levels of the object tree use jatos.groupSession.find. Gets the object from the locally stored copy of the session and does not call the server.

- *@param {string} name* - name of the field
- *@return {object}* - the value that is stored under name

Example: If the internal Group Session is `{"a": 1000, "b": "watermelon"}`

```
// Since the parameter is the key's name and not a path it doe
s not start with a "/"
var b = jatos.groupSession.get("b"); // b is "watermelon"
var c = jatos.groupSession.get("c"); // c is undefined
```

Example: With `jatos.groupSession.get` you can only access the first level of the object tree - if you want another level use `jatos.groupSession.find`. If the internal Group Session is `{"a": {"a1": 123, "a2": "watermelon"}}`

```
var a1 = jatos.groupSession.get("a1"); // a1 is undefined !!!
var a = jatos.groupSession.get("a"); // a is { "a1": 123, "a
2": "watermelon" }
```

`jatos.groupSession.set(name, value, onSuccess, onFail)`

Like `jatos.groupSession.add`, but instead of a JSON Pointer path it accepts a name of the field to be stored. Therefore it works only on the first level of the Group Session's object tree. If the name already exists in the Group Session the value will be overwritten.

- *@param {string} name* - name of the field
- *@param {object} value* - value to be stored
- *@param {optional callback} onSuccess* - Function to be called if this patch was successfully applied on the server and the client side
- *@param {optional callback} onError* - Function to be called if this patch failed
- *@return {jQuery.deferred.promise}*

Example: If the internal Group Session is `{"a": 1234}`

```
// Since the parameter is the key's name and not a path it doe
s not start with a "/"
var b = jatos.groupSession.set("b", "koala");
```

then after the Group Session is successfully updated the new internal object is `{"a": 1234, "b": "koala"}`.

Since there is a slight chance that the session update was not successful it's a good idea to provide callback functions for both cases. To provide success or fail callback functions you can either specify the onSuccess/onError parameters or use jQuery' deferred object.

Example with jQuery's defered:

```
var deferred = jatos.groupSession.set("b", "koala");
deferred.done(function () {
  alert("Group Session was successfully updated");
});
deferred.fail(function () {
  alert("Group Session synchronization failed");
});
```

`jatos.groupSession.getAll()`

Returns the complete Group Session data (might be bad performance-wise). Gets the object from the locally stored copy of the session and does not call the server.

- *@return {object}*

Example:

```
var groupSession = jatos.groupSession.getAll();
```

`jatos.groupSession.setAll(value, onSuccess, onFail)`

Replaces the whole session data. If the replacing object is rather large it might be better performance-wise to replace only individual paths. Each session writting involves sending the changes in the session via a JSON Patch to the JATOS server. If the session is large this data transfer can take some time. In this case use other session functions, like 'set', 'add', or 'replace'.

- *@param {object} value* - value to be stored in the session
- *@param {optional callback} onSuccess* - Function to be called if this patch was successfully applied on the server and the client side
- *@param {optional callback} onError* - Function to be called if this patch failed
- *@return {jQuery.deferred.promise}*

Example:

```
var o = {"a": 123, "b": "foo"};
jatos.groupSession.setAll(o); // Overwrites the current Group S
ession with the object o
```

Since there is a slight chance that the session update was not successful it's a good idea to provide callback functions for both cases. To provide success or fail callback functions you can either specify the onSuccess/onError parameters or use jQuery' deferred object.

Example with jQuery's defered:

```
var o = {"a": 123, "b": "foo"};
var deferred = jatos.groupSession.setAll(o);
deferred.done(function () {
  alert("Group Session was successfully updated");
});
deferred.fail(function () {
  alert("Group Session synchronization failed");
});
```

`jatos.groupSession.clear(onSuccess, onFail)`

Clears the whole Group Session data and sets it to an empty object `{}`.

- *@param {optional callback} onSuccess* - Function to be called if this patch was successfully applied on the server and the client side
- *@param {optional callback} onError* - Function to be called if this patch failed
- *@return {jQuery.deferred.promise}*

Example:

```
jatos.groupSession.clear();
```

Since there is a slight chance that the session update was not successful it's a good idea to provide callback functions for both cases. To provide success or fail callback functions you can either specify the onSuccess/onError parameters or use jQuery' deferred object.

Example with jQuery's defered:

```
var deferred = jatos.groupSession.clear();
deferred.done(function () {
  alert("Group Session was successfully updated");
});
deferred.fail(function () {
  alert("Group Session synchronization failed");
});
```