How to load data from an experiment

The data can be loaded into SPSS / R / Excel / LibreOffice / OpenOffice directly after running an experiment!

All the experimental data is saved to a sub-folder of \chattool\data\Saved experimental data.

The name of the sub-folder is xxxxname, where xxxx is a prefix that is a unique number that is automatically generated by the chat server. The sub-folder that has the highest number is the most recent experiment. The sub-folder that has the second-highest number is the second most recent experiment…etc.

The second part of the name of the folder is the name given to the particular experiment settings (don't worry too much about this…the default name setting is perfectly fine. If you do want to change it, you will need to change the getDefaultSettings() method in the ConversationController object of the experiment you are running)

The folder will always have (at least) 6 files in it. The most important file, that contains all the chat turns ready for analysis is "turns.txt"

Turns.txt

99% of the time you will only need to use **turns.txt**. (see below for the other files and what they contain) This is the file that contains

(1) each turn that is typed

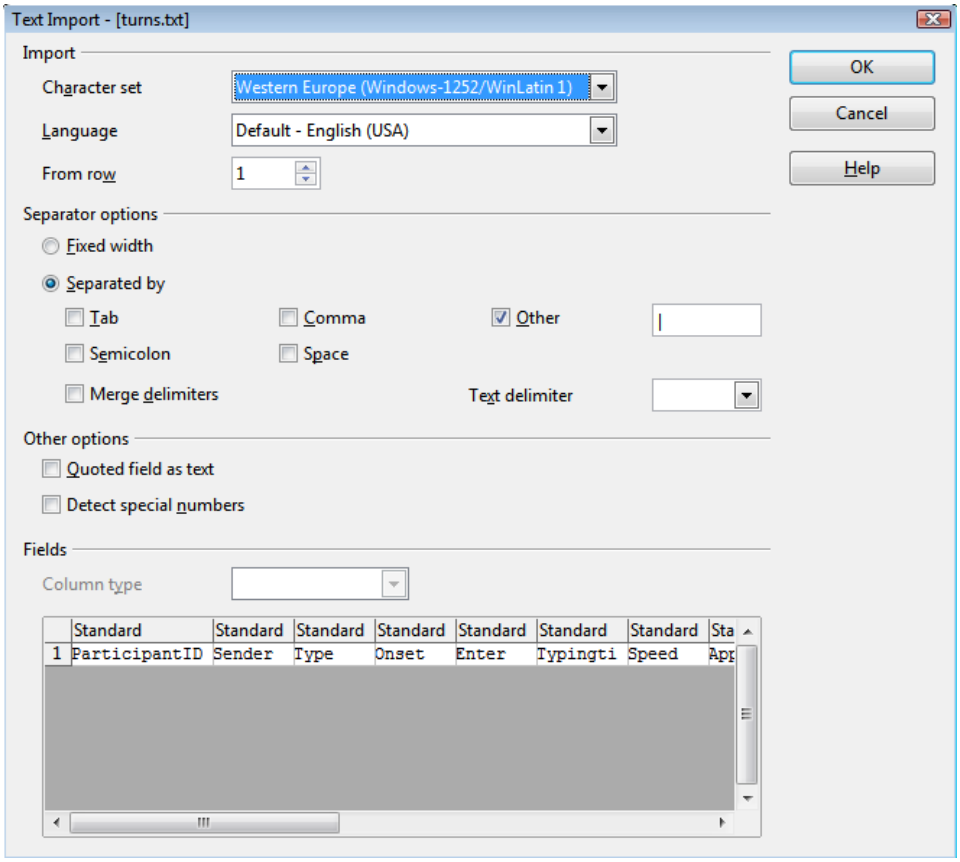
(2) each individual keypress

(3) Any additional data that you save from your ConversationController object with c.saveData(String s).

This file is stored as a CSV file, but with the "|" character (ignore the speech marks as a delimiter. This file opens very straightforwardly in OpenOffice. / Excel. See below for what to select when using OpenOffice. Note that every separator option is unchecked, apart from "Other", which is "|".

Important:

The chat tool is configured to save all text output in UTF-8 format.



The spreadsheet is arranged vertically.

Each row stores information associated with events that occur:

1. **Turns.** Whenever a participant presses ENTER or SEND and transmits a turn to the other participant(s), the turn is saved on a row. See the next section (below) to see what information is stored in each column. All chat text turns have a value of type = "Default" (third column in the spreadsheet)

2. **Keypress**. Whenever a participant presses a key, this keypress information is also instantly transmitted to the server, and is also stored on a separate row. Keypress information has a value if type = "Keypress" (third column in the spreadsheet)

3. **Other data.** Some interventions also save intervention-specific data in the spreadsheet. If you want to save an additional row of data you can make use of diet.server.Conversation.saveData(…) in your program.

The column headers below describe the information that is stored for "normal" chat text. These are turns that have Type = "Default" (see third column)

ParticipantID is the ID number the participant entered to log in to the chat server.

Sender is the Username (nickname) that the participant chose when logging into the chat server.

Type specifies what kind of information is being saved. If this value is "Default", this means the row contains a "normal" chat text turn. If the row contains "KEYPRESSS" this contains the information from a single keypress.

ClientTime records the exact timestamp (As recorded on the **client**) when the participant pressed ENTER.

Onset is the time when the participant typed the first character of the turn. For keypress data, "onset" is the time of the individual keypress.

Enter is the time when the participant pressed the SEND button or pressed, as the name suggests, ENTER.

TypingTi stands for "typing time" and is simply Enter – Onset.

Speed is the typing speed in characters per second

AppOrig is the "apparent origin" of the turn. This is who the turns appears to have been sent by (this is important when conducting experiments that "spoof" the identity of the person who sent the message.

Text is the text that was typed by the participant (ie the text that was typed followed by pressing SEND or RETURN

Recipients This is a list of all the participants who received the text that was sent.

Blocked. This records whether, during turn formulation, the chat tool interface was blocked by the server: Some experimental setups require the interface of a participant's chat client to be blocked by the server, preventing text entry (for example when the chat server engages the other participant in an artificial subdialogue). If a participant's chat client has been blocked while formulating a turn, then the data recorded for that turn should typically be discarded from analysis. To allow this, the this column will have the value "Blocked" if turn formulation has been blocked.

DDels contains the number of times the participant deleted text in the private turn formulation window

KDels contains the number of times the participant pressed the delete key (KDels and DDels are typically the same, but occasionally differ – for example if a participant selects a word that is 20 characters long and presses delete once, this would lead to KDELS = 1 and DDels = 20.

DIns records the number of edits performed on the text in the private turn-formulation window that occur at no turn-terminal positions. For example if someone types HELO and then edits that to HELLO, this would lead to DIns = 1.

DDels*N and **DIns * N** multiply each occurrence by the distance from turn terminal position at which they occur. (This is an attempt to score edits that occur further back with higher scores – essentially measuring editing cost – we've hardly used these measures)

PriorTurnByOther_TimestampOnClientOfReceipt: Because computer networks might have network latencies (ie. Lags) this can mean that all participants might not see the same messages at the same time. Consequently measuring response times / turn-taking intervals on the server can result in noisy data, depending on how severe the variation in

network lag is. To get round this problem the next three measures are used to record participants' response times on their own clients – These measures show what the turn is in response to (and how long the interval was). N.B. Of course, unless supplemented with eyetracking, there is no way of being sure that the participants were actually responding to the previous turn. If the immediately preceding turn was produced by another participant, the measure **PriorTurnByOther_TimestampOnClientOfReceipt** records the timestamp of receipt of that immediately preceding turn. If the immediately preceding turn was produced by the same participant who created the current turn, this value is empty. This information can be used to calculate the exact time interval between receipt of the message and the first keypress (effectively a measure of response time / reaction time / turn-taking time).

Reaction time = **PriorTurnByOther_TimestampOnClientOfReceipt** - **Onset**

PriorTurnByOther_ApparentUsername: This measure records the (apparent) username of the participant who produced the immediately preceding turn. In normal chat text where there is no intervention the apparent username will be the same as the actual username. But in interventions that spoof the identity of the participants, the apparent username will be the spoofed username.

PriorTurnByOther_Text: this measure records the text of the immediately preceding turn.

Errorlog.txt

This is a file that contains error messages caused by Exceptions that weren't caught by your own code – it's a good idea to always check this file to see if it's nonzero.

This file can also sometimes contain error messages that were sent from the clients- this method, though is not so reliable, as errors on the clients are typically to do with connectivity, and if the connection is broken,causing the error, no error message can be sent.

Turns.dat

This contains a copy of all the turns typed by the participants AND all the artificial turns generated by the server. The format of this file is in Java's Java's native serialized object format. If you want to read in this file, you can create an ObjectInputStream in Java and associate it with that file, and then simply read in every single object. So far we've never needed to do so, so there's no example that makes use of it – but it should be very straightforward to do so.

Note that this file will NOT be the same as turns.txt – it will not contain the keypress information, nor will it contain any additional data that you save with c.saveData(String s). This is primarily because the ConversationHistory data is stored on the computer as turns – the additional info that you save as c.saveData(String s) is not

Expparameters.xml

This is a copy of the experimental Template containing the parameters that were used to start the experiment. This is to ensure that you know what settings were used for the particular experiment (allowing you to rerun it, or even to restart it after a crash). To open this file use a text editor with no "linewrap". (It's not actually an xml file – at the time of programming it, Java's xml implementation was buggy, so we had to implement it as text). See the section on Parameters for more information..

Messages.dat

This is a file that contains every single message sent and received by the server (see the class diet.Message) – the format of this file is in Java's native serialized object format – if you wanted to debug what happened, you can create an ObjectInputStream in Java and associate it with that file, and then simply read in every single object. So far we've never needed to do so, so there's no example that makes use of it – but it should be very straightforward to do so.

Messages.txt

This file is a text-based, filtered version of Messages.dat – it doesn't include all the fields of each message, but does store basic information such as the recipients and the timestamp. Load it in a spreadsheet using the same method as using turns.txt.