z-Tree 2.0

Zurich
Toolbox For
Readymade
Economic
Experiments

What’s new in Version 2.0

Urs Fischbacher

1 A table that survives the end of a treatment: The `session` table

Consider the following experiment: First you conduct a public goods treatment. The income earned in this public goods treatment can then be used in a second treatment, e.g. in a market game. In version 1.1 of z-Tree, you have to define one treatment that includes both the public goods game and the market game. This is so, because tables were deleted after the end of each treatment (of course, after the tables were saved). In this new version there is a new table, the `session` table, that survives the end of a treatment. This table has - as the `subjects` table - one record per subject. With the `session` table you can easily solve the above problem. In the first treatment, you store the total profit in a variable `Income` in the `session` table. Because the `session` table is not deleted after the end of the treatment, you can use the stored value in the second treatment.

How do you store data in the `session` table? In programs in the `session` table, the corresponding record of the `subjects` table can be accessed with the scope operator. Therefore, in your first treatment, you can enter the following line into a program of the `session` table:

```
        Income = : Total Profit;
```

How do you retrieve data from the `session` table? You can access variables in the `session` table as in any other table. There is only one problem: Z-Tree always checks that you do not use variables that are not defined earlier in the treatment. Therefore, you have to declare which variables in the `session` table you will use in a treatment. All tables are listed in the stage tree at the beginning of the background. A double click at the `session` table opens a dialog. In this dialog you enter all variables you want to use in the field "Used variables", separated by a comma.[1]

2 User defined tables

In z-Tree 2.0 it is possible to define more tables than the standard tables `subjects`, `globals`, `summary`, `contracts` and `session`. You may want to use this feature for different purposes:

[1] The procedure that checks the correctness of programs needs to know which variables are defined. In the session table we cannot define variables with an assignment statement (which is the general method) because an assignment would override the value.

- Multiple contract tables for different markets.
- Random period payment: A table contains an array with the profits made in each period. You can use this table for more than one treatment, i.e. you can select a random period out of the periods of different treatments.
- Storing a discrete function.
- Using a profit calculator[2] and an auction simultaneously.

To use a new table, you have to define it at the beginning of the background with the menu item "New Table..." in the "Treatment" menu. A dialog opens where you can enter the properties of the table.

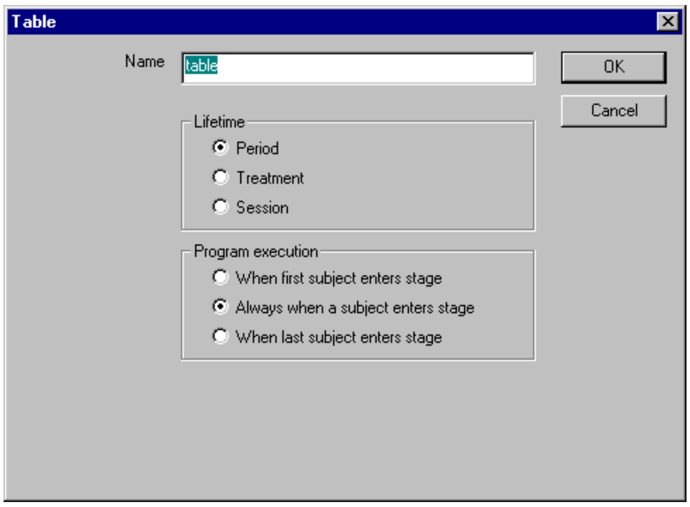


Table
Name
table
OK
Cancel
Lifetime
Period
Treatment
Session
Program execution
When first subject enters stage
Always when a subject enters stage
When last subject enters stage

Lifetime: If a table's lifetime is "**Period**", the table is reset whenever the end of a period has been reached, i.e., all records are deleted (of course after they have been saved). The `subjects`, `globals`, and `contracts` tables have a lifetime of one period. As in the `subjects`, `globals`, and `contracts` tables, the data of the previous period of a table `t` can be accessed with `OLD`, i.e., in the table called `OLDt`.

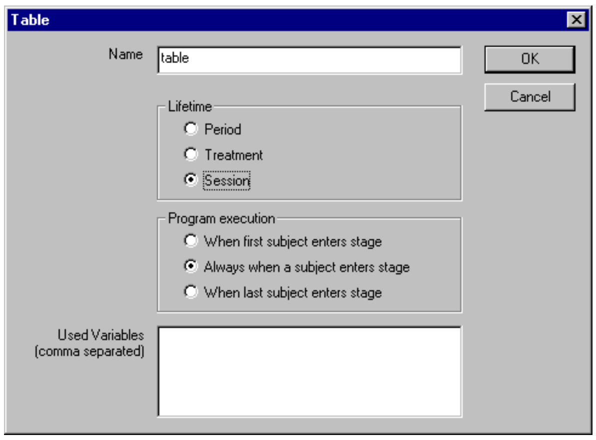
Tables with lifetime "**Treatment**" are reset after the end of the treatment. They do not change after the end of a period.

Tables with lifetime "**Session**" are never reset. Nevertheless they are stored in the xls file at the end of each treatment. To be able to access variables defined in another treatment (that was executed before) you

[2] In a profit calculator, a subject can enter its own strategy and the strategy of its opponent. The calculator then calculates the profits of the two subjects. You store the calculations in a contracts table and present it in a contracts list box.

have to declare the variables in the field "Used Variables" as explained in the chapter on the `session` table.

Remark: All tables are saved at the end of each period. This allows to recover a session from the end of the last period.



Table

Name: table

Lifetime: Period / Treatment / [✓ Session]

Program execution: When first subject enters stage / [✓ Always when a subject enters stage] / When last subject enters stage

Used Variables (comma separated):

Program execution: If the subjects do not simultaneously enter a stage, then there are three possibilities when programs are executed. In z-Tree this option is defined in the table definition: Programs in a table can be executed (i) whenever a subject enters, (ii) only when the first enters the stage, or (iii) only when the last subject enters the stage. For instance, the programs in the `globals` table are executed with the first subject, programs in the `summary` table with last subject. Programs in the `subjects` and in the `session` table are executed for the subject who enters this stage.

Remark: In programs executed in user defined tables you can use an owner variable as in the `contracts` table.

3 The Dutch auction and postponed program execution

In a Dutch auction you need a flexible clock. You must be able to define the steps of the price decrease as well as the time interval between two price decreases. You can do this with a `later` statement. If you start with a price of 1000 and every 3 seconds you want do decrease the price by 10, you write the following program (for instance into the `globals` table):

```
Price = 1000;
later ( 3 ) repeat {
     Price = Price – 10;
}
```

The `later` statement has the following general form:

```
later( expression ) repeat { statements }
```

1. The expression is evaluated. Let t be the value of the expression.
2. If t is greater or equal to 0, then after t seconds, first, the statements are executed and then we go to 1.
3. If t is smaller than zero nothing happens anymore.
4. When a subject leaves a stage, the `later` statements generated in this stages are terminated.

Example:

```
Delay = 10;
Price = 100;
later( Delay ) repeat {
     Price = Price –12;
     if ( Price <= 0 ) {
           Price = 0;
           Delay = -1;
     }
}
```

In this example the following happens: First, `Delay` is calculated. The value is 10, i.e., positive. Hence, 10 seconds later, 12 is subtracted from `Price` and `Delay` is calculated again. The value is still 10, i.e., positive and therefore the statements in the curly brackets are executed in a second step. As long as `Price` is positive, i.e. during the first 8 steps, nothing else happens. In the ninth step `Price` is first changed from 4 to –8. Then, `Price` is set to zero and `Delay` is set to –1. Now, `Delay` has the value of minus one, i.e., it is negative. Therefore, no further `later` statements are executed.

There is a second form of the `later` statement. It applies to situations where you want the statements to be executed at most once. In this case, you write:

```
later( expression ) do { statements }
```

Also in this case the expression is evaluated. If the value t is negative, nothing happens. If the value is greater than or equal to zero, the program waits until the corresponding number of seconds is over and then the statements in the curly brackets are executed once.

Application 1: Trial periods for an auction experiment

Assume, you plan to conduct an experiment with a double auction. Because the user interface will be rather difficult, subjects should be able to learn the handling. This is best achieved with some trial periods. A problem with a trial period is that there is already interaction between the subjects. Thus, uncontrolled learning takes place before the actual experiment starts. To allow for controlled learning you can simulate the entries of the other subjects in an auction. To do this you enter a sequence of `later` statements as the following example shows:

```
later ( 8) do { contracts.new{ Seller = 12; Buyer = -1; Price = 12; }}
later (13) do { contracts.new{ Seller =  3; Buyer = -1; Price = 15; }}
later (21) do { contracts.new{ Seller =  7; Buyer = -1; Price = 17; }}
later (34) do { contracts.new{ Seller =  5; Buyer = -1; Price = 18; }}
```

Application 2: Double auction with an external shock

Suppose you want to change the economic environment during an ongoing double auction. Let p be the parameter you want to change from a value of 100 to 50 after one minute. You write the following program:

```
P=100;
later ( 60 ) do {
     P=20;
}
```

4 New crash security concept

The crash security in z-Tree 1.x was based on GameSafe and on some files where the profits were saved. In practice, whenever the server crashed, z-Tree was restarted using the profit files and the GameSafe was only used when clients had to be restarted. In z-Tree 2.0, there are more tables and therefore the profit files are no longer sufficient to describe the state of the experiment at the end of a period. This was one reason to reorganize the recover procedure. Another reason was the instability of the restart procedure with GameSafe.

After a crash of z-Tree, you now have to do the following:

1. Restart z-Tree.
2. Open the clients' table.
3. Restart all clients with the Menu "Run/Restart All Clients". If some clients do not connect, go to the client and reconnect the client manually. To reconnect a client manually, you quit z-Leaf with <alt>-[F4] and start z-Leaf again.

   If *no* client connects then you have four possibilities:

   - You restart all clients manually.
   - You wait a short while and try to restart the clients later. (It can take up to 4 minutes.)
   - You shut down and restart the experimenter PC. (Exit windows and logon again is not sufficient.)
   - You start z-Tree on another computer.
4. With the Menu "Run/Restore Client Order" you sort the clients in the same order as they have been in the crashed session.
5. With the Menu "Run/Reload database" you restore all tables. Since the tables are stored after each period you can restore the state of z-Tree just before the crashing period.
6. Check how may periods have been played. You find this information for instance in the `summary` table or in the `subjects` table.
7. Open the treatment you were running before. If n periods have been played, set the number of practice periods to –n (minus n).
8. With the Menu "Run/Start treatment" you start the treatment again.

5 Configurable Directories

In version 2.0 you can use command line options to set the directories where data are stored. These options are very helpful if you perform many sessions and want to organize the data. The following options are available:

`/xlsdir`	Sets the directory where you want the xls file to be stored.
`/sbjdir`	Sets the directory where you want the subjects file to be stored.
`/zttdir`	Sets the directory where you want the treatment and questionnaire files to be stored automatically (before they run).
`/datadir`	Sets the directory where you want the xlsdir, the sbjdir, and the zttdir to be stored.
`/adrdir`	Sets the directory where you want the address file to be stored.

`/paydir`	Sets the directory where you want the payment file to be stored.
`/privdir`	Sets the directory where you want the address and the payment file to be stored.
`/gsfdir`	Sets the directory where you want the GameSafe to be stored.
`/tempdir`	Sets the directory where you want the temporary files @lastclt.txt, @db.txt and @prevdb.txt to be stored.
`/leafdir`	Sets the directory where you want the file server.eec to be stored.
`/dir`	Sets the directory where you want all of the above mentioned files to be stored.
`/homedir`	Sets the home directory, i.e. further directories are defined relative to this home directory.

The commands in the command line are processed one after each other. Therefore the command

```
ztree /datadir x /sbjdir y
```

sets the subjects directory to y whereas

```
ztree /sbjdir y /datadir x
```

sets the subjects directory to x, i.e., the command line option `/sbjdir y` is overridden.
Remark: You can create a shortcut if you use a combination of command line options more than once.

Tip: Create a leaf directory.

Put z-Leaf into a directory different from the directory where z-tree is located. The subjects get only access to this directory. This prevents subjects from getting access to your experimental data.

If you set the `leafdir` option in z-Tree, then z-Tree puts its address into the file server.eec in this leaf directory. Hence, the z-Leaves there can start automatically, i.e., they automatically connect with z-Tree.

Tip: Create a testing shortcut for z-Tree

First make a directory named "testtrash". Then make a shortcut

```
ztree /dir testtrash
```

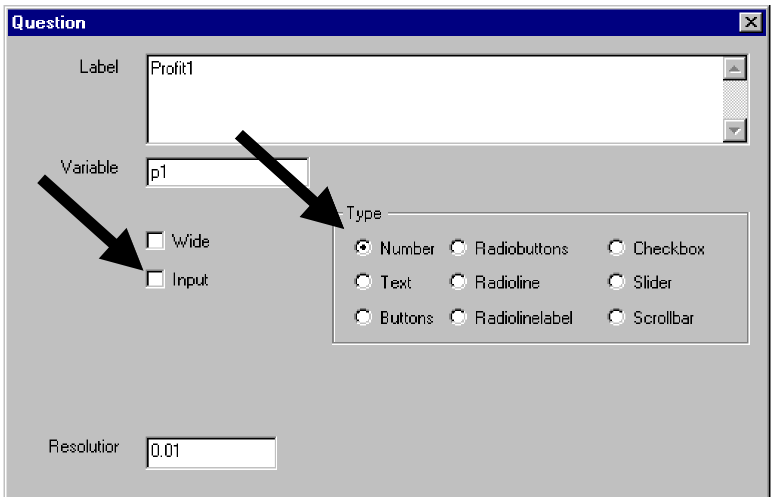
Use this shortcut whenever you perform tests. Use a different shortcut when you actually run the session. After you conducted all tests, you can delete all the files in the directory testtrash without risking to delete important files.

Tip: Data protection: Separate non anonymous data

Save all the experimental data in a database which can easily been read by the researchers at your institution. For data protection reasons you should store the critical data, i.e. the non-anonymous files (address file and payment file) in a different place. The access to this place should be much more restricted.

6 More flexible output in payment file

Sometimes you need more output in the payment file. Consider for instance the case where in an individual decision making experiment each subject gets paid for her decisions in a randomly selected period. Suppose furthermore that you want to use a physical random device for this selection. For that you need more information in the payment file than just the sum of all profits, i.e., you need the profits per period. By adding questions to the address screen, you can add any variable of the `session` table to the payment file. If you want the variable `p1` to appear in the payment file with a precision of two digits and you want the corresponding column to be labeled with "Profit1", you have to make the following entries in the question dialog:


Question
Label: Profit1
Variable: p1
Wide [ ] Input [ ]
Type: [✓ Number] Radiobuttons Checkbox Text Radioline Slider Buttons Radiolinelabel Scrollbar
Resolutior: 0.01

Note in particular that you must set the type to "Number" and the "Input" option must be unselected.

7 Using the payment file for mail merge

Instead of just printing the payment file you can use it to produce individual receipts with the mail merge function of your word processor. In the following, we explain the procedure step by step (for MS Word).

Before the Session starts.

- Make a mail merge document. You can use the fields "Subject", "Computer", "Interested" "Name" and "Profit" and the variables you added to the payment file. (The name of the mail merge variable is the label of the variable, not the name of the variable itself.)
- You can open the word processor and open the mail merge document.

When the payment file is written.

- Open the mail merge document (and bring it to the front).
- Open "Mail Merge" (Seriendruck) in the menu "Extras".
- Select in point 2 of the mail Merge manager the option "Open Data Source" and select the payment file. You find it in the directory where you started z-Tree or in the directory you defined with the `privdir` option. The easiest way to find it is to view the directory in details mode and sort it according to "Modified".
- Select point 3 in the dialog to generate the merged document.
- Check whether the document is correct and print it.

8 Question forms can be omitted

Suppose, you have conducted an experiment where subjects played in different roles, for instance as proposers and responders, and you want to use different questionnaires about motivations for the different types. In the new version of z-Tree, you can *omit* question forms. Hence, if you want to present two different questionnaires to two different types of subjects, you put the question forms for both types into one questionnaire document and omit the forms selectively for the type you want.

The procedure to omit question forms is similar to the procedure to omit stages in a treatment: You set the variable `Participate` to zero, if you want the particular subject not to fill out that question form. Since there is no `subjects` table available when you run a questionnaire, the variable `Participate` has to be set in the `session` table. You enter the program that sets the variable `Participate` in the question form dialog.

Important: Do not forget to copy the type variable (the variable that determines whether the subject is a proposer or a responder) into the `session` table.

9 Default entry in items

An input item can now have a default entry. This entry is used if you show an initial value (option "show value") or if you allow not to fill a field (option "empty allowed").

In general, you can set a default value also in a program. However, this is impossible in contract creation boxes.

10 Option for buttons: Clear entry also for contract selection boxes

The " Clear entry" option now works in contract creation boxes as well as in contract list boxes.

11 Checkers can be used for warnings

Checkers can contain two buttons, a "yes" button and/or a "no" button. The buttons that are non-empty appear in the dialog where the message is displayed. There are three cases:

- There is only a "no" button: This is the normal case: The message is displayed. The subject can click at the button and the input is *not* accepted.
- There is only a "yes" button: This generates a warning: The message is displayed. The subject can press the button and the input *is* accepted.
- Both buttons are non empty. In this case the subject can press either one of the two buttons. If he or she presses the "yes" button, the input is accepted - if he or she presses the "no" button, the input is not accepted.

12 Function gettime()

The function `gettime` returns the time in seconds since the startup of the computer. You get the time since the start of treatment with the following program (in the `globals` table):

```
TimeAtTreatmentStart = 0;
TimeAtTreatmentStart = if( Period == 0, gettime(),
                       OLDglobals.find( TimeAtTreatmentStart ) );
TimeSinceTreatmentStart = gettime() - TimeAtTreatmentStart;
```

13 Handling cases with `elsif`

Consider a situation where you have 4 types of players and for the different types you have different profit functions. You can program this with a nested `if` statement as follows:

```
if ( type == 1 ) {
     // first calculation
}
else {
     if ( type == 2 ) {
           // second calculation
     }
     else {
           if ( type == 3 ) {
                 // third calculation
           }
           else {
                 // fourth calculation
           }
     }
}
```

It is not easy to keep track of all the necessary closing brackets. With `elsif`, there is an easier way to express the above program:

```
if ( type == 1 ) {
     // first calculation
}
elsif ( type == 2 ) {
     // second calculation
}
elsif ( type == 3 ) {
     // third calculation
}
else {
     // fourth calculation
}
```

14 Loops: `while` and `repeat`

In z-Tree 2.0, you can program loops. There are two forms:

```
while( condition ) { statements }
```

The condition is evaluated and as long as the condition is `TRUE`, the statements are executed and the condition is reevaluated.

```
repeat { statements } while ( condition );
```

The statements are executed. Then the condition is evaluated and as long as the condition returns `TRUE`, the statements are executed again and the condition is reevaluated. With the repeat statement, the statements between the curly brackets are executed at least once.

Remark: Loops are frequently a source of program bugs: If the condition in a loop never returns `FALSE`, this loop runs forever. Bugs in loops are particularly bad because you have to quit z-Tree to stop the loop.

15 Italian Version

Z-Leaf can now also be started in Italian language.

16 Smaller improvements

- If possible, buttons in z-Leaf are big enough to display the whole label.
- In contract list boxes, the color of selected items is the same as the one chosen in the color control panel.
- The time when buttons are pressed by subjects is now stored even if no header is present.
- In general, a document can be closed without warning if it has not been changed since it has been saved the last time. In z-Tree 1.1, treatments can be closed without warning even if only an *automatic* save has been conducted. In z-Tree 2.0, the warning appears as long as the file has not been saved with a save *command*.
- There is now a check to ensure that history box items have unique labels. (Otherwise only one column would be displayed.)

- If there is a syntax error in a program, z-Tree reports the line where the error was found.
- "Preferred Workstation" in the role parameter dialog is eliminated.
- In the stage tree after a "cut" command: The selection is set to the element where the item was removed from, i.e., a "paste" just after a "cut" inserts an element where it has been before.
- Input array variables can have expressions in the index.

17 **Fixed Bugs**

In this chapter you find the bugs that are fixed in version 2.0. This will be most interesting if you had a problem and you want to check whether this problem is now solved. In this list, the problem is reported, not the change. So, new in version 2.0 is the fact that all the listed bugs are removed.

- Crash: Select period parameter, then any matching command.
- If an expression is entered into the max field of an item the maximum is equal to zero.
- At some places, the option value in the layout field cannot be a variable.
- Checks with OLD-tables in combination with `new` do not work.
- Crash if treatment is started with disconnected client.
- Input array variables cannot contain spaces.
- Contract creation box with no items cannot be filled.
- In integrated variables "<<" does not work.
- Error messages in z-Leaf: If the label is formatted with RTF, the error displays the RTF-code.
- `same( Group )` does not work in the condition in a contract list box.
- Auction after auction deactivates time display.
- In questions of type number the checks do not work.
- Crash with ":" in layout fields as in "`!text: 1=one; 2:two; `".
- Import has trouble with \n.
- The function `roundup` makes error if the second argument is different from 1.
- Windows list is not updated after "Save as".