

**z-Tree** <sup>2.1</sup>

Zurich  
Toolbox For  
Readymade  
Economic  
Experiments

## Reference Manual

Urs Fischbacher

# Table of content

7.1	The Stage Tree Elements .....	5
7.1.1	The background .....	5
7.1.2	Parameter Table .....	7
	Period Parameters .....	7
	Role Parameters .....	7
	Specific Parameters.....	7
7.1.3	Screen .....	7
7.1.4	What can be contained in stage tree elements ?.....	8
7.2	Menu Commands.....	9
7.2.1	File Menu.....	9
	New Treatment .....	9
	New Questionnaire .....	9
	Open.....	9
	Close .....	9
	Save .....	9
	Save As.....	9
	Export Treatment.....	9
	Export Questionnaire.....	9
	Export GameSafe.....	9
	Export Table.. .....	9
	Import... .....	10
	Import Table... .....	10
	Page Setup... .....	10
	Print... .....	10
	Previous files .....	10
	Quit .....	10
7.2.2	Edit Menu .....	10
	Undo .....	10
	Cut .....	10
	Copy.....	10
	Paste.....	10
	Copy groups.....	11
	Paste groups .....	11
	Insert cells.....	11
	Remove cells.....	11
	Find.....	11
	Find Next .....	11
7.2.3	Treatment Menu.....	11
	Info.....	11
	New Stage.....	11
	New Table.....	13
	New Program .....	14
	New Box .....	15

	New header box .....	17
	New standard box .....	17
	New calculator button box .....	18
	New history box .....	18
	New help box .....	19
	New container box .....	19
	New grid box .....	20
	New contract creation box .....	21
	New contract list box .....	23
	New contract grid box .....	24
	New message box .....	25
	New button .....	26
	New checker .....	27
	New item .....	28
	Stage Tree .....	30
	Parameter table .....	30
	Check .....	30
	Import variable table .....	30
	Show variable... .....	31
	Append variable .....	31
	Append text .....	31
	Matching: Partner .....	31
	Matching: As First Period .....	31
	Matching: Stranger .....	31
	Matching: Absolute Stranger .....	31
	Matching: Absolute Typed Stranger .....	32
	Matching: Transform .....	32
	Utilities: Make Stage Names Unique .....	33
	Language .....	33
7.2.4	Questionnaire Menu .....	33
	Info .....	33
	New Address Form .....	33
	New Question Form .....	34
	New Ruler .....	34
	New Question .....	34
	New Button .....	36
	Check .....	36
	Language .....	37
7.2.5	Run Menu .....	37
	Clients' Table .....	37
	Shuffle Clients .....	38
	Sort Clients .....	38
	Save Client Order .....	38
	Start Treatment .....	38
	Start Questionnaire .....	38
	xyz Table .....	38
	Stop Clock .....	39

	Restart Clock .....	39
	Discard a client .....	39
	Leave Stage.....	39
	Stop after this period.....	39
	Restart all clients.....	39
	Restore Client Order .....	40
	Reload Database .....	40
7.2.6	Tools Menu.....	40
	Separate Tables.....	40
	Join *.sbj File.....	40
	Join files.....	40
	Merge Files .....	41
	Fix File.....	42
7.2.7	Windows Menu.....	42
	Treatment and questionnaire files.....	42
	Toolbar.....	42
	Status Bar.....	42
7.2.8	Help Menu .....	42
7.3	Programming environment.....	43
7.3.1	Programming language.....	43
	Error handling.....	43
	Constants .....	43
	Operators .....	43
	Statements.....	44
	Functions .....	45
	Table functions .....	46
	Iterator .....	47
	Syntax diagram of the programming language.....	47
	Definitions .....	47
	Terminal elements .....	47
7.3.2	Scope environment .....	51
7.3.3	Program execution.....	52
7.3.4	The tables and their standard variables.....	52
	The subjects table .....	52
	The globals table.....	53
	The summary table .....	53
	The session table.....	54
	The contracts table.....	54
	The table of the previous period “OLDtables” .....	54
	User defined tables .....	54
7.4	Text formatting.....	55
7.4.1	Variable output in test display .....	55
7.4.2	Formatting with RTF commands known to be processed .....	56

7.4.3	Combining RTF and inserted variables .....	57
7.5	Command line options.....	57
7.6	The Import Format .....	59

## 7.1 The Stage Tree Elements

This chapter explains options of the stage tree elements that cannot be created. The stage tree element that can be created are explained in the Menu section in the menu where the element is created.

### 7.1.1 The background

Information not specific for a particular stage may be inserted and defined in the “Background”. First, the background contains a list of tables that are used for calculations. Then, in the screens of the background, elements can be placed that are used in all stages. In our example, all active screens have a header window and all waiting screens a text “Please wait until the game continues”. The programs of the background are run at the beginning of a period. They are used for defining constants. In the background itself, i.e., when you double-click it, some central parameters of the treatments can be viewed and changed:

**Number of subjects:** This number is constant for a whole session. Therefore all treatments must have the same number of subjects.

**Number of groups:** Number  $g$  of groups to which the subjects are assigned. This means that the subjects belong to a group with an ID between 1 and  $g$ . This value is used for the matching commands and for checking.

**# practice periods:** Number of periods in which no profit is paid out. The number entered can be negative. In this case the period counter starts later. If, for instance, the number of trial periods is  $-2$  and the number of paying periods is 10, then the treatment extends from period 3 to period 10.

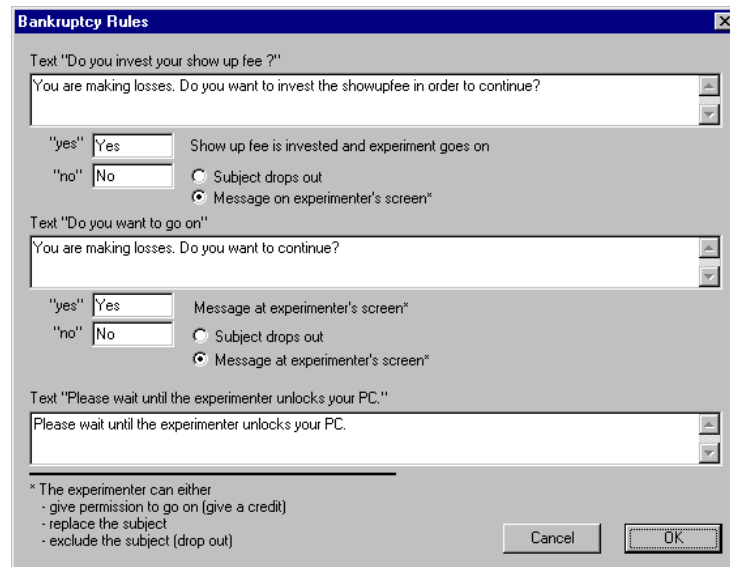
**# paying periods:** Number of periods in which the profit is paid out.

**Exch. rate [Fr. / ECU]:** How many Franks (\$, DM, £) are paid out per "experimental currency unit" [ECU]

**Lump sum payment [ECU]:** Amount in internal unit that is credited at the beginning of period 1 *of the treatment*. This amount figures in the variable TotalProfit.

**Show up fee [Fr.]:** Amount credited to the subjects at the start of the session. Only the show-up fee of the first treatment played is decisive.

**Bankruptcy rules:** By clicking this button the parameter of the bankruptcy rules may be adjusted. In the tutorial, you get a detailed description about how to deal with losses.



The dialog box titled "Bankruptcy Rules" contains three sections, each with a text area and two buttons labeled "yes" and "no".

- Section 1:** Text "Do you invest your show up fee ?". Text area: "You are making losses. Do you want to invest the showupfee in order to continue?".
  - "yes" button: "Yes" (text is present).
  - "no" button: "No" (text is present).
  - Options: ☒ "Show up fee is invested and experiment goes on", ☐ "Subject drops out", ☐ "Message on experimenter's screen".
- Section 2:** Text "Do you want to go on". Text area: "You are making losses. Do you want to continue?".
  - "yes" button: "Yes" (text is present).
  - "no" button: "No" (text is present).
  - Options: ☐ "Message at experimenter's screen", ☐ "Subject drops out", ☒ "Message at experimenter's screen".
- Section 3:** Text "Please wait until the experimenter unlocks your PC.". Text area: "Please wait until the experimenter unlocks your PC.".

At the bottom, there is a note: "\* The experimenter can either  
- give permission to go on (give a credit)  
- replace the subject  
- exclude the subject (drop out)".

Buttons: "Cancel" and "OK".

The first question appears when the subject has suffered losses that can be covered by the show-up fee. The second message appears when the losses suffered by the subject exceed the show-up fee.

The fields near "yes" and "no" contain the labeling of buttons. If both buttons are not empty, the user has a choice. If one of the two buttons is empty, the user can only confirm the given answer. Depending on what button the user has been clicked, the action stated beside the button is executed.

**Show-up fee is invested and the game continues:** The show-up fee is used to cover the losses and the game continues.

**The player drops out automatically (OUT):** By putting the subject into the state OUT, he or she is taken out of the game. This is not wholly unproblematic - the programs need to be equipped for this process. You can, for example, set an input variable to an impossible value, in order to be able to discern that a subject has dropped out.

**Message to the experimenter:** No message appears at the client's screen. The experimenter sees in the state of the subject that she is in the losses.

### 7.1.2 Parameter Table

#### Period Parameters

**Prompt:** If a text is entered here, it is displayed on the server before the start of the period. The treatment will only continue when OK is clicked on the server.

**Program:** This program is run in the globals table after the programs in the background, the specific parameter program and the role program have been run.

#### Role Parameters

**Name:** of no consequence

**Preferred work station:** At the start of the session this role is assigned to the subject whose machine has announced itself under this name. This option is only of significance for the first treatment of a session.

**Program:** This program is run in the subjects table, after the programs in the background and the specific parameter program have been run.

#### Specific Parameters

**Group:** Group number of the subject in this period.

**Name:** of no consequence

**Program:** this program is run in the subjects table after the programs in the background have been run.

### 7.1.3 Screen

Only one option is available for screens: "Background screen is being used". If this field is marked, the windows of the background are taken over and placed first.

In every stage there are two screens, the active screen which is shown to the subject who participate in this stage until they have concluded it. The waiting screen is shown when subject wait for the next stage.



### 7.1.4 What can be contained in stage tree elements ?

In the following, we list for all stage tree elements what kind of other elements they can contain. We use the notation "a < b c" if the element a contains the element b followed by c. A star after an element means zero or one or more instances one of this kind of element. The first line therefore signifies that the stage tree contains one background followed by some stages (zero or more). The notation "a = b | c" means that the term a is one of b or c. This notation is used to define the term box.

stagetree	< background stage*
background	< table* program* activescreen waitingscreen
stage	< program* activescreen waitingscreen
activescreen	< box*
waitingscreen	< box*
box	= standardbox   headerbox   helpbox   containerbox   gridbox   calculatorbuttonbox   historybox   contractcreationbox   contractlistbox   contractgridbox   messagebox
standardbox	< item* button*
headerbox	< -
helpbox	< -
containerbox	< box*
gridbox	< item* button*
calculatorbuttonbox	< -
historybox	< item*
contractcreationbox	< item* button*
contractlistbox	< item* button*
contractgridbox	< item* button*
messagebox	< -
item	< -
button	< checker* program*

## **7.2 Menu Commands**

### **7.2.1 File Menu**

#### ***New Treatment***

A new treatment is created. It contains the background and the background screens.

#### ***New Questionnaire***

A new questionnaire is created. It is completely empty.

#### ***Open...***

Opens a treatment or a questionnaire. A dialog, in which the treatments are shown, appears. In order to open a questionnaire, you have to select the corresponding option.

#### ***Close***

Closes the front most window.

#### ***Save***

Saves a treatment or a questionnaire.

#### ***Save As...***

Saves a treatment or a questionnaire under a new name or at another place.

#### ***Export Treatment...***

Treatment is written in textual form in a file. A treatment exported in this way may also be read in again.

#### ***Export Questionnaire...***

Questionnaire is written in textual form in a file. A questionnaire exported in this way may also be read in again.

#### ***Export GameSafe...***

A GameSafe is translated from an unreadable binary format into a gigantic ASCII file.

#### ***Export Table..***

A table is written in a tab-separated file.

***Import...***

Imports a treatment or a questionnaire.

***Import Table...***

A table (tab-separated file) is read into the current window, if it is a table window.

***Page Setup...***

(Not yet implemented)

***Print...***

(Not yet implemented)

***Previous files***

Opens a treatment or questionnaire previously used.

***Quit***

Quits the program. A warning message appears if the session is still running, i.e., if a treatment is running, if a questionnaire is running or if the payment file is not yet written.

**7.2.2 Edit Menu*****Undo***

(Not yet implemented)

***Cut***

Deletes the marked branch in the stage tree or in the questionnaire and places it onto the clipboard.

***Copy***

Copies onto the clipboard the marked area of a table or the selected branch in the stage tree or in the questionnaire.

***Paste***

In the parameter table: Copies a table from the clipboard and pastes it at the selected position.

In the stage tree or in the questionnaire: Copies the branch from the clipboard and pastes it at the selected position.

### ***Copy groups***

Copies the groups of the selected area of the parameter table onto the clipboard.

### ***Paste groups***

Copies the groups from the clipboard and pastes them at the selected position in the parameter table.

### ***Insert cells***

Adds more rows or columns immediately after the last row or column selected. If more than one row or column are selected, a corresponding number of rows or columns is added.

### ***Remove cells***

Removes complete rows or columns. Complete rows are selected by clicking at the small cell at the left border of the parameter table. Complete columns are selected by clicking at the cell at the top of the parameter table.

### ***Find...***

(Not yet implemented)

### ***Find Next***

(Not yet implemented)

## **7.2.3 Treatment Menu**

### ***Info...***

Opens a dialog in which the parameters of the selected element can be viewed and changed.

### ***New Stage***

A stage corresponds a screen that a subject has to go through. It actually consists of the two screens, the active screen the subject goes through and the waiting screen that is shown when the subject leaves the stage and waits for the next stage.

In the following, we show the stage options.

**Name:** Name of the stage. Used for documentation and as a name for the state in the clients' table.

**Start: Wait for all:** Subject may not enter this stage if not all subjects are at least in the waiting state of the preceding stage.

**Start: Start if possible:** No condition for subjects to enter this stage.

**Start: Start if:** A condition can be given that must be satisfied before subjects can enter the stage. This condition is calculated in the subjects table. (Be careful that this condition is eventually satisfied. Otherwise the experiment will never continue.)

### Number of Subjects in Stage

**At most one per group in stage:** If checked only one subject per group can be in this stage. This option is used in posted offer markets.

**... and in previous stage(s):** If the checkbox is checked, this stage and the preceding stage form a set. If also in the previous stage this box is checked also the stage before belong to this set and so forth. Now, only one subject per group may be in any stage of such a set.

### **Leave Stage after timeout**

**If no input:** The stage is terminated after timeout if no input is expected.

**Yes:** The stage is terminated after timeout even if input is expected.

**No:** The stage is not terminated after timeout.

**Timeout:** The amount of seconds after which the timeout occurs. This expression is calculated in the globals table.

In the stage tree, the stage options are displayed after the name. The following notation is used:

<b>Start: Wait for all:</b>	=
<b>Start: Start if possible</b>	-
<b>Start: Start if</b>	( <i>condition</i> )
<b>Not At most one per group in stage</b>	=
<b>At most one per group in stage</b>	-
<b>... and in previous stage(s)</b>	-...-
<b>Leave Stage after timeout, If no input:</b>	( <i>timeout</i> )
<b>Leave Stage after timeout, Yes</b>	( <i>timeout</i> )A
<b>Leave Stage after timeout, No</b>	( <i>timeout</i> )N

### **New Table**

A table stores input and calculations. Tables are accessed with their name.

**Name:** Name of the table.

**Lifetime:** For how long can this table be uses.

**Lifetime Period:** The table is reconstructed after each Period. As in the subjects or globals table, you get access to the data of the previous table with the prefix OLD in front of the table name.

**Lifetime Treatment:** The table is available during the treatment. At the end of the treatment it is destroyed (after being stored it of course).

**Lifetime session:** As the session table, the table survives the end of the treatment.

**Program execution:** You can specify whether programs defined for the table at the beginning of a stage are execute when the first subject enters the stage, when the last enters or Always when a subject enters that stage.

**Used Variables:** For table with lifetime session, you can define what variable are uses. This allows to use them in a program with having to define them within

### ***New Program***

In programs, calculations can be carried out. Programs modify the content of tables.

The image shows a dialog box titled "Program". It has a "Table" dropdown menu currently showing "subjects". To its right is an "Owner Variable" text field. Below the "Table" dropdown is a "Condition" text area. Below the "Condition" text area is a "Program" text area. On the right side of the dialog, there are "OK" and "Cancel" buttons.

**Table:** The table in which the program is executed. For details see Scope environment.

**Owner variable:** Allows accessing a record in the subjects table with only the scope operator. It is the record its value of the Subjects variable equals the value of the owner variable in this table.

Example: You have a variable Buyer in the contracts table which contains the subject ID if the Buyer. By putting Buyer into the Owner variable of the program for the contracts table, you can access the record of the buyer in the subjects table just with the scope operator.

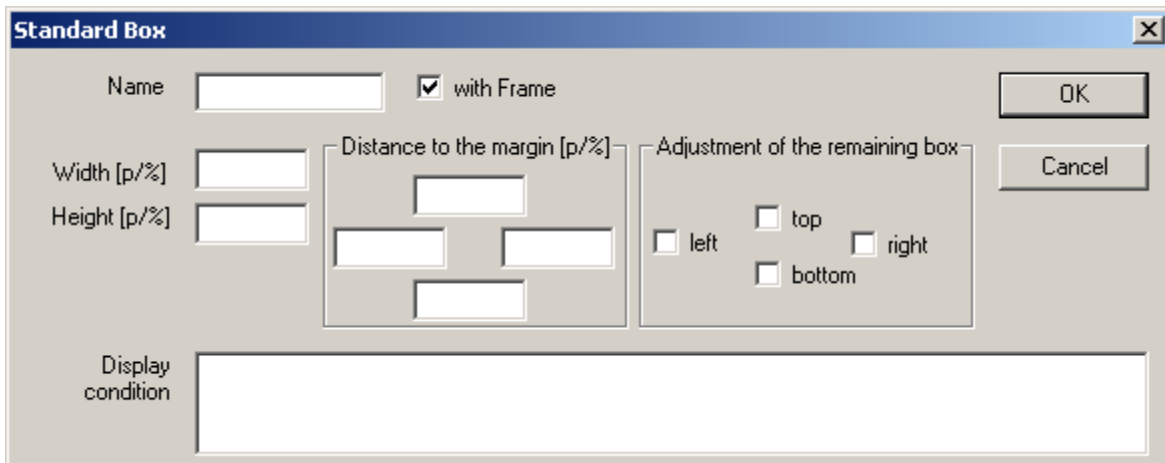
**Condition:** Condition for which records the program is executed.

**Program:** The program that is executed.

### **New Box**

The subjects' screens are made up of boxes. These are rectangular parts of the screens. The boxes are positioned within the "remaining box" in the order in which they appear in the stage tree. In the beginning, the "remaining box" constitutes the whole screen. Later the "remaining box" is adjusted according to the definition of the boxes. The dialog for a box is explained below.





The image shows a 'Standard Box' dialog box with the following fields and controls:

- Name:** A text input field.
- with Frame:** A checked checkbox.
- Width [p/%]:** A text input field.
- Height [p/%]:** A text input field.
- Distance to the margin [p/%]:** A group box containing three text input fields for top, left, and bottom margins.
- Adjustment of the remaining box:** A group box containing four checkboxes: left, top, right, and bottom.
- Display condition:** A large text area.
- Buttons:** 'OK' and 'Cancel' buttons.

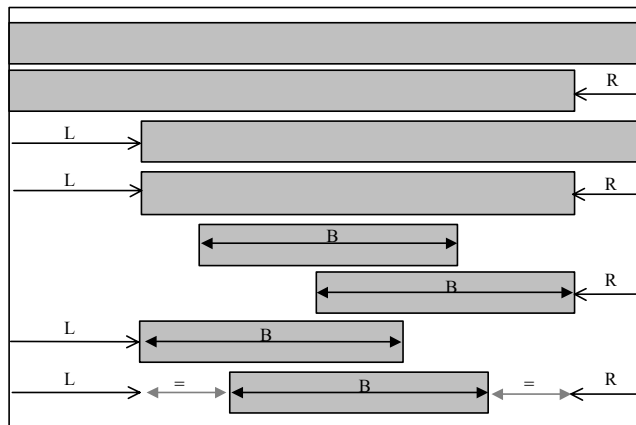
**Name:** Name of the box. Used for documentation only.

**With frame:** Is there a frame around the box or not.

**Width:** Width of the remaining box in pixels [p] or percent [%]; optional.

**Height:** Height of the remaining box in pixels or in percent; optional.

**Distance to the margin:** Indicates how far away the margin is from the margin of the remaining box; in pixels or in percent of the remaining box; optional.



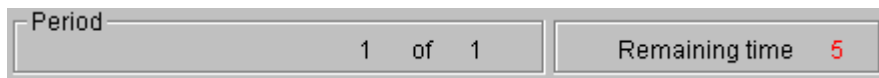
**Adjustment of the remaining box:** Where is the remaining box cut-off, if at all?

Width, height and distance to the margin are optional. Depending on which fields have been filled, the box is placed within the remaining box. The graph below shows all cases for the fields width (W), distance to left margin (L) and distance to right margin (R).

**Display condition:** If this field is nonempty, the box will only be shown if this condition is satisfied. The condition is calculated in the subjects table. It is reevaluated whenever data is changed, i.e. this condition can be used to turn on and off boxes dynamically. (If the field is empty, the box will always be shown.)

### ***New header box***

Creates a header box.



Period number and time are displayed in a header box. The following options can be adjusted:

**Show current period number:** Period number is displayed or not.

**Show total number of periods:** Total number of Periods is displayed or not.

**Name of "period":** Name in front of the period number. "Period" is used as standard. However, "Trading day" or "Round", for example, are also possible.

**Term for "out of":** At "3 out of 10" (Periods).

**Prefix for trial periods:** "Trial" is standard.

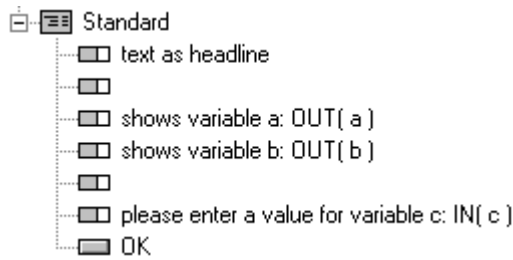
**Display time:** Is time being displayed to the subjects? If not, not even a time message is generated.

**Term for "Remaining time":** "Remaining time [sec]:" is standard. If this field is empty, no time is displayed. In this way you can do without displaying the time and yet ask the subjects to continue when time has run out.

**"Please make your decision now":** Text that appears when the allotted time has run out.

### ***New standard box***

Creates a standard box.



Variables of the subjects table may be displayed or entered. The items are displayed from top to bottom. The window is divided into a label column (left) and a variable column (right). The variables are always displayed in the variable column. The label always appears in the left column if the variable is defined or if it consists only of an underscore (“\_”). If the variable is empty, the label is regarded as title and written centered over the whole window. If the label consists of more than one line, it is aligned to the left.

**Button position:** Buttons can be placed into corners or centered at a margin.

**Button arrangement:** If more than one button is defined in this box, the further buttons can be placed either in rows or in columns. If they are placed "in rows", z-Tree first tries to fill a row starting from the corner given by the button position. If the first row is full, a next row is filled starting from the same corner.

### ***New calculator button box***

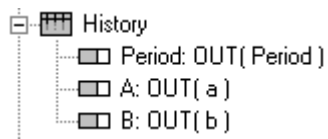
Creates a button box.



Defines a button by which the Windows NT Calculator can be called up. It serves as a substitute for subjects who have forgotten their calculator.

### ***New history box***

Creates a history box.



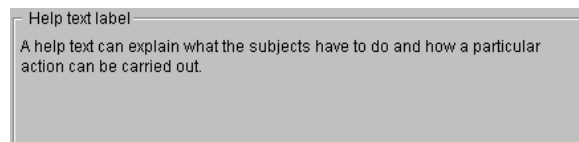
Period	A	B	
3	36	12	
4	45	24	
5	51	21	
6	37	11	
7	63	28	

The results from previous periods are listed in a table. A label row contains the labels. If the table is too long, a scrollbar appears. The current period appears at the end of the table. The scrollbar is adjusted in such a way as to make this line visible at the beginning.

**Show also current period:** If checked all periods including the current period are shown. If it is not checked, only the previous periods are shown.

### ***New help box***

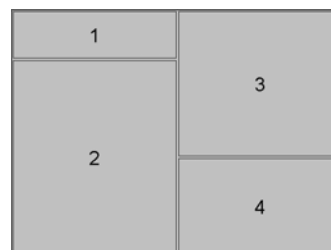
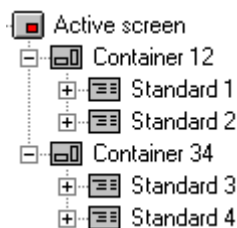
Creates a help box.



A help box is a box with a label that contains a text. A label and a help text can be entered. If the help text is too long to appear in the area reserved for the help box, a scrollbar appears.

### ***New container box***

Creates a container box, i.e., a box which may contain other boxes.



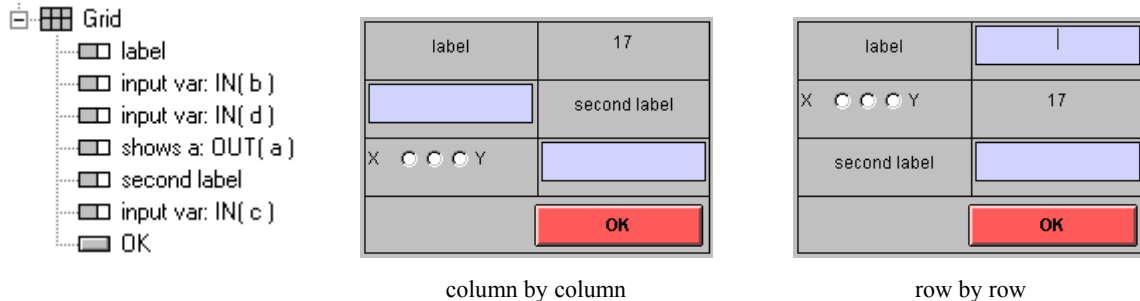
If you want to directly place the boxes in the figure above, it becomes necessary to choose the appropriate sizes. By defining container boxes you can easily make sure that the boxes always match one another. Container boxes are boxes that can contain other boxes.

In the example we define six boxes, the boxes 1,2,3, and 4, as well as two container boxes, 12 and 34. We define the width in the container box 12 and cut-off the remaining box at left. Now container box 34 fills the region on the right. In boxes 1 and 3 we define the heights and cut the remaining box off at the top. In this way we have not entered a size specification at more than one place, and even if we should change something, the appearance of the screen remains the same.

If we move a box from one place to another inside the stage tree, it is moved *after* the box where the mouse button is released. By moving a box over the *icon* of a container box it is moved *into* the container box and positioned at its beginning.

### New grid box

Creates a grid box, a box that shows data of the subjects table in a tabular form.



The items are displayed in a table. Each item belongs to one field. If the item contains a variable, this variable is displayed. During checks the label is used to convey to the subject in which field he or she has made a mistake. The label is only displayed if the item does not contain a variable.

The individual options are:

**Num. rows:** Number of rows of the grid, including label row if it exists.

**Num. columns:** Number of columns of the grid, including label column if it exists.

**Input row-by-row:** The items are filled into the grid row-by-row. By pressing the tab key the subject can make the entry row-by-row.

**Input column-by-column:** The items are filled into the grid column-by-column. By pressing the tab key the subject can make the entry column-by-column.

**First row contains labels:** First row is label row. Only used for “Separate labels by lines”.

**Height [of the first line]:** Height of the first line in % of the other lines.

**First column contains labels:** First column is a label column. Only used for “Separate labels by lines”.

**Width [of the first column]:** Width of the first column in % of the other columns.

**Separate labels by lines:** If the first row is a label row, a dividing line is drawn between the first and the second row. The same procedure applies for the first column.

**Separate rows by lines:** Lines are drawn between the rows.

**Separate columns by lines:** Lines are drawn between the columns.

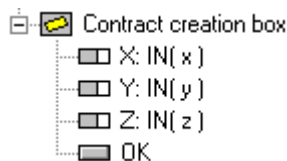
**Button position:** Buttons can be placed into corners or centered at a margin.

**Button arrangement:** If more than one button is defined in this box the further buttons can be placed either in rows or in columns. If they are placed in rows, z-Tree first tries to fill a row starting from the corner given by the button position. If the first row is full, a next row is filled starting from the same corner.

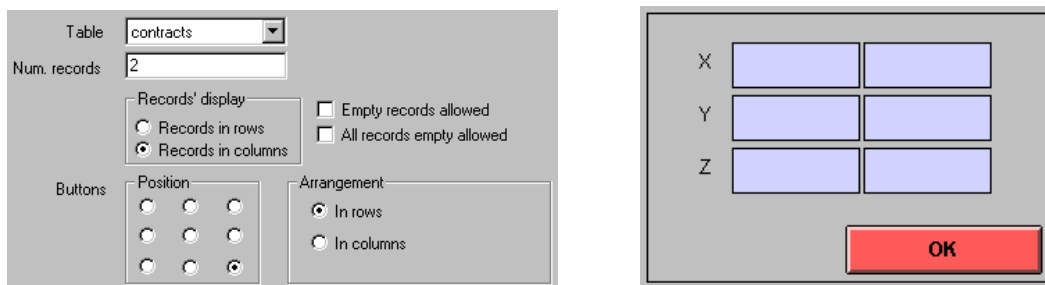
### ***New contract creation box***

Creates a contract creation box.

We show in an example how a subject can enter two new contracts in a contract creation box. A contract should contain the values x, y, and z. Therefore, we put three input items, x, y and z as well as an OK button into the contract creation box. In the stage tree the contract creation box looks like this:



How the contract creation box looks like in z-Leaf can be modified in the box' dialog:



option in z-Tree

box in z-Leaf

When you click the button, z-Tree checks if all entries have been made. Next the checkers in the button are checked, and if these are OK, two new contracts are added to the contracts table. Finally, the programs of the button are run. If we are in an auction stage, further entries can be made, otherwise the stage is concluded. You may specify the following options:

**Table:** All contract boxes can be linked to any one of the tables. This paves the way for future extensions. At the moment only the contracts table is feasible.

**Num. records:** Number of records that have to be entered. Only one record is feasible in an auction stage. You can also enter a variable from the subjects table here, for example a constant, numContracts. In this way you can define treatments in which subjects have to enter different numbers of contracts.

**Records' display:** The records can be displayed in columns or in rows. In the above example, we show the two options.

**Empty records allowed:** If this option is marked, the subject can fill out less records than are displayed. For each record, however, either all fields must be filled or none. In this way you can allow the subjects to offer less than the maximum possible number of contracts.

**All records empty allowed:** All records can be left empty which means that no record is created.

**Button position:** Position of the button (the buttons) in the box.

**Button arrangement:** When the first button lies in the lower right hand corner, horizontal means that further buttons are added on the left, until one has to begin a second row. The first button in the second row lies above the first button. The other cases are analogous.

### New contract list box

x	y	z
38	35	38
13	53	52
21	36	49
18	21	40
28	55	44

t:

u:

OK

In a contract list box a selection of records is listed. Which records appear as well as the order of the records in the list can be set. Contract list boxes are continually updated. Therefore, whenever a subject effects a change in the contracts table, this is immediately registered and displayed in the contract list box.

The contract list box is composed as follows: The values of the output items are entered in one list, each record on a separate row. If there is not enough space for all contracts in the list, a scrollbar appears. If there are input items, these appear at the bottom of the list in the same layout as in a standard box.

When a contract list box contains a button, the subject can select a record and fill in the input items. When the button is clicked and the checks are OK, the variables of the input items are set and the programs are run for the selected record.

It is useful for the button of a contract list box to contain a program. Only with the help of a program can the contracts table be changed in such a way as to show that the contract has been selected and who has selected it. Such a program could be the following:

```
Acceptor = :Subject;
```

Acceptor is a variable of the selected records. “:Subject” is the variable Subject in its own record in the subjects table. (For details see chapter on programs below).

On the options:

**Table:** All contract boxes can be linked to any one table. This paves the way for future extension. At the moment, however, only the contracts table is feasible here.

**Owner var.:** Owner variable is a variable of the contracts table. The records for which the value of this variable equals the variable Subject are shown blue for each subject. Example: Seller is the variable in the contracts table in which the subject number of the seller is written. When all sales offers are displayed on



the auction screens of the sellers, one can show one's "own" contracts in blue. These are the contracts for which the variable seller has the value Subject.

**Condition:** Condition for the records displayed.

**Sorting:** Expressions according to which sorting is carried out, separated by semi-colons. When sorting is to be undertaken in descending order, the expression is preceded by a minus sign.

Example: a; -b; -c; sorts the columns in the following way:

a	1	3	6	6	6	7	8	8	8	10	10
b	9	2	7	5	5	4	4	4	2	6	3
c	7	2	4	5	3	2	8	4	5	3	3

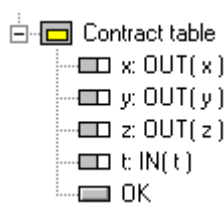
**Scrolling:** After each change to the list scrolling to the beginning or to the end takes place.

**Mark best foreign contract:** If no other contract is selected, the contract in the direction of scrolling is automatically selected, i.e., if scrolling has taken place towards the end, the last foreign contract is chosen. A contract is considered as an *own* contract if the owner variable has the value of the subject ID of this subject. All other contracts are foreign contracts. If there is no owner variable, all contracts are considered as foreign.

**Button position:** Position of button (buttons) inside the box.

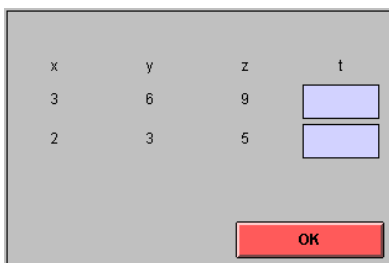
**Button arrangement:** When the first button lies in the lower right hand corner, horizontal means that further buttons are added at left until one has to begin a second row. The other cases are analogous.

### New contract grid box

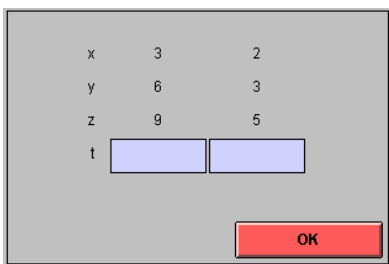


Contract table

- ☐ x: OUT( x )
- ☐ y: OUT( y )
- ☐ z: OUT( z )
- ☐ t: IN( t )
- ☐ OK



records in rows



records in columns

The selected records are shown in a table. Values of the output items and input fields for the input items appear in this table. There must be space for all contracts in the table.

This box works similarly to the standard box, except that variables from the contracts table, not from the subjects table can be viewed and changed in it. The images above show contract grid boxes. To the left, the records are in the rows and, to the right, in the columns.

On the options:

**Table:** All contract boxes can be linked to any one table. This paves the way for future extensions of z-Tree. At the moment, however, only the contracts table is feasible here.

**Owner variable:** The owner variable is a variable in the contracts table. For each subject, the records for which this variable has the same value as the variable Subject are shown in blue.

**Condition:** Condition for the records that are displayed. The expression in this field is evaluated for each record. Those records are shown for which the value of the condition is TRUE.

**Sorting:** Expressions according to which sorting is carried out, separated by a semi-colon. If sorting is to be carried out in descending order, the expression is preceded by a minus.

**Records' display:** As in the contract creation box.

**Empty records allowed:** If this option is marked, the subject can fill out less records than are displayed. For each record, however, either all fields must be filled or none.

**All records empty allowed:** All records can be left empty.

**Button Position:** Position of button (buttons) inside the box.

**Button Arrangement:** When the first button lies in the lower right hand corner, horizontal means that further buttons are added at left until one has to begin a second row. The other cases are analogous.

### ***New message box***

Creates a message box. If there is a message box, messages generated by z-Tree are not displayed in separate windows. This makes the restart of a client easier because in case of a restart also all message reappear and have to be confirmed with a mouse click.

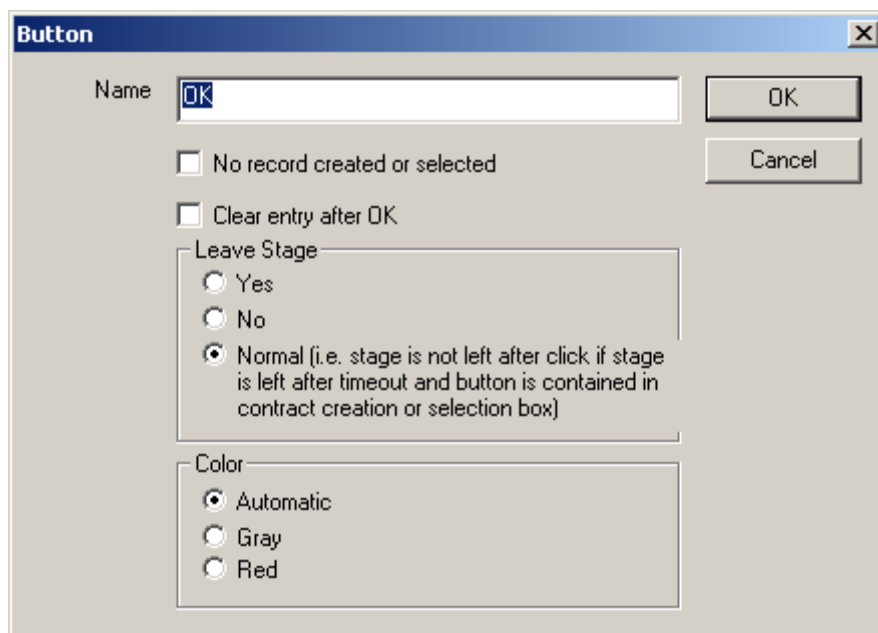
**Initial Message:** Appears when the stage starts.

## **New button**

Input is concluded by clicking a button. The labeling of the button can be defined. If the text consists of an underscore (“\_”), no button appears. However, space for a button is kept free. If the button label contains the “&”-character, the next character is underlined. If you want to insert an “&”-character into the button label, you have to duplicate it.

Buttons can contain checkers and programs. Checkers are used for checking subjects’ entries. Programs in buttons are executed when the button is clicked and when all checks are OK.

Standard boxes, grid boxes and all contract boxes can contain buttons.

The image shows a 'Button' configuration dialog box. It has a title bar with the word 'Button' and a close button. Inside, there is a 'Name' field with 'OK' entered. To the right of this field are 'OK' and 'Cancel' buttons. Below the 'Name' field are two checkboxes: 'No record created or selected' and 'Clear entry after OK', both of which are currently unchecked. Below these checkboxes is a section titled 'Leave Stage' containing three radio buttons: 'Yes', 'No', and 'Normal (i.e. stage is not left after click if stage is left after timeout and button is contained in contract creation or selection box)'. The 'Normal' option is selected. At the bottom is a section titled 'Color' with three radio buttons: 'Automatic', 'Gray', and 'Red'. The 'Automatic' option is selected.

**Name:** Label of the button. If this consists only of an underscore (“\_”), no button appears. However, the space for the button is kept free.

**No record created or selected:** If a button in a contract creation box has checked this option, then pressing the button will not create a new record. If a button in a contract list box has checked this option, then pressing the button will not select a record, i.e., this button can be pressed even if no records is selected.

**Clear entry after OK:** After the button has been pressed, the fields belonging to this button are cleared. This option is useful to prevent subjects from making the same entry repeatedly.

**Leave Stage:** Per default, stages are left, if the button is in a normal box or if the stage is not an auction, i.e., it waits for input to conclude. The options here allow overriding this behavior.

**Color:** Button which are related to input appear in red other buttons in gray. This option allows to use the color more flexibly.

### ***New checker***

Checkers are used for more complicated input checks. If a condition is not met, a message is displayed

**Condition:** Condition that has to be satisfied. The message appears if the condition is not satisfied. In contract creation boxes, checkers are run for *each new* record. In contract list boxes, they are run for the *selected* record. In all other boxes, the check is carried out in the subjects table.

**Message:** Message that appears if condition is not satisfied. If there is a message box, the message appears in this box. Otherwise, a dialog appears that needs to be confirmed by the subject.

**"yes"-Button:** Button for meaning accept the input nevertheless.

**"no"-Button:** Button for meaning do not accept input.

If there is only a "yes"-button, then the message appears and after pressing this button the input is accepted. If there is only a "no"-button, then the message appears and after pressing this button the input is rejected. If there is a "yes" AND a "no"-button, then the message should contain a question. Pressing the YES button accepts the input pressing NO rejects it.

### **Note on how checks are performed**

First of all we always check if each field is filled. The subject needs to enter a value into each field, unless it is explicitly stated in the item that the field may be empty.

A minimum value and a maximum value need to be entered in the definition of items. If a number has to be entered, the resolution is also given in the layout field. The subjects' entries must lie between the minimum and the maximum value and it must be a multiple of the resolution. If this is not the case, the entry is not accepted. If the label is not empty, the subject receives a message telling him/her which error has occurred in which field. If the label is empty, it bleeps and the field containing the error is marked. No message appears.

If all these check are passed, the checkers in the button are handled.

## ***New item***

Items are used for displaying and reading in variables. An item contains the name of the variable and information on how to display it. We call an item "input item", if the checkbox "Input" is checked. In this case subjects have to make an entry. We call an item "output item", if the checkbox "Input" is not checked. In this case a variable will be displayed. Items can be contained in standard boxes, grid boxes, history boxes and all contract boxes.

**Label:** Name of the variable displayed to the subjects. This text may also be empty.

**Variable:** Variable that is displayed or read in. In the case of input items only variables or array variable with a number as index are allowed. (Array variable may not contain spaces and the index must be written with the number of digits given by the increment of the array.) For output items any expression is allowed. If the variable is empty, the item only displays the label. If the variable contains only an underscore (" \_"), the layout is the same, as when a variable has to be entered, except that no value for the variable is displayed.

**Layout:** If a number, a variable or an expression is entered here, the value of that field determines how the variable is rounded when it is displayed. If for instance 0.2 is entered in layout, and the variable contains the value 12.34, then 12.4 is displayed. If it is an input item, then the number is used to check whether the number entered by the subject is a multiple of this number.

An exclamation mark makes further formatting options available:

```
!text: value1 = label1 ; value2 = label2 ; ...
```

In the case of an output item the text *label<sub>i</sub>* appears when the variable has the value *value<sub>i</sub>*. In the case of an input item it is the other way round, i.e. if a *text<sub>i</sub>* is entered, the variable gets the value *value<sub>i</sub>*. If text different from all *text<sub>i</sub>*s is entered, an error message appears.

```
!radio: value1 = label1 ; value2 = label2 ; ...
```

Labeled radio buttons appear. The value of the variables again corresponds to the values, the labels of the buttons are the corresponding labels.

```
!radioline: leftvalue = leftlabel; rightvalue = rightlabel; number
```

A row of radio buttons appears. They may be accompanied by labels to the left and to the right. Leftvalue is the value of the button at the left margin. Rightvalue is the value of the button at the right margin. The value of number determines the number of buttons.

```
!slider: leftvalue = leftlabel; rightvalue = rightlabel; number
```

A slider appears. The slider may be accompanied by a text on the left and on the right. Leftvalue is the value of the variables at the left margin. Rightvalue is the value of the variables at the right margin. The value of number determines the resolution, i.e., the number of possible slider positions.

```
!scrollbar: leftvalue = leftlabel; rightvalue = rightlabel; number
```

A scrollbar appears. The scrollbar may be accompanied by a text on the left and on the right. Leftvalue is the value of the variables at the left margin. Rightvalue is the value of the variables at the right margin. The number sets the resolution, i.e., the number of possible scrollbar positions.

```
!button: value1 = label1 ; value2 = label2 ;...
```

Buttons appear, arranged similarly to radio buttons. By clicking a button the variable is set to the corresponding value. Besides this, each button also has the same effect as the OK button. Hence, at most one variable per screen can have the button option. For output items, the button options yields the same display as the text option.

```
!checkbox: 1 = text;
```

A checkbox appears. This checkbox is labeled with text. If the box is marked, this corresponds to the value 1. If it is not marked, this corresponds to the value 0.

**Input:** If this checkbox is marked, the subject has to enter a variable. Otherwise the variable is displayed.

**Show Default:** This option appears only with input variables. Normally an input field is empty in the beginning. If this option is chosen, the input field is filled with the current value from the database.


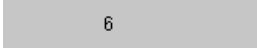










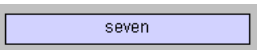



**Empty allowed:** If this option is marked, the variable may be left empty.

**Default:** This option appears only with input variables. If a field is left empty, this value is taken as input instead.

**Minimum:** The minimum permitted value for input variables.

**Maximum:** The maximum permitted value for input variables

## Examples of item layouts

Layout	input variable	output variable
2		
!radio: 1 = "86.8"; 24 = "102.8";		
!radioline: 0="zero";5="five"; 6;		
!slider: 0 ="A"; 100= "B"; 101;		
!scrollbar: 0="L";100= "R";101;		
!checkbox:1="check me";		
!text: 1= "one"; 2 = "two"; 3 = "three"; 4 = "four"; 5 = "five"; 6 = "six"; 7 = "seven"; 8 = "eight"; 9 = "nine"; 10 = "ten";		
!button: 1 = "accept"; 0 = "reject";		

## Stage Tree

Opens the window with the stage tree.

## Parameter table

Opens the window with the parameter table.

## Check

Checks the programs and parameters for errors.

## Import variable table...

You give the name  $v$  of a variable and choose a text file that contains a table. Then the line

$$v = w;$$

is added into the program of the specific parameters. Here w stands for the value from the table. If the format of the table does not correspond with the format of the parameter table, the specific parameters are filled where there is a value in the table.

### ***Show variable...***

Sets the name of a specific parameter to the value of a variable. So, this value becomes visible in the parameter table.

### ***Append variable...***

Append the value of a variable to the name of a specific parameter.

### ***Append text ...***

Append the some text to the name of a specific parameter.

### ***Matching: Partner***

A partner design for group matching is selected: The first subjects constitute group 1, the next, group 2, etc., for each period. If a range of the parameter table is selected, group matching is only carried out for this region.

### ***Matching: As First Period***

Groups are matched for all periods in the same way as they are now matched for the first period. If a region of the parameter table is selected, group matching is only carried out for this period. The first period of the selection counts as the first period.

### ***Matching: Stranger***

Group matching is carried out at random for each period. If a region of the parameter table is selected, group matching is only carried out for this region.

### ***Matching: Absolute Stranger***

Group matching is carried out in such a way that each subject is never in the same group with any other subject more than once. This is only possible for a limited number of periods. In the periods in which such a group matching is no longer possible, 1 is set as group number for all subjects. If a region of the parameter table is selected, group matching is only carried out for this region.



Caution: The execution of this command can take up some time. Depending on the given number it can require milliseconds, minutes or days. Before the command is executed, a warning appears and you get the possibility to cancel the execution of the command.

### **Matching: Absolute Typed Stranger**

This creates an absolute stranger matching for heterogeneous groups: If, for example, a group consists of a company and two employees, we often do not want the same subject to have different roles. This means that only those groups that contain a company and two employees are allowed.

Caution: The execution of this command can take up some time. Depending on the given number it can require milliseconds, minutes or days. Before the command is executed, a warning appears and you get the possibility to cancel the execution of the command.

In the following table the number of periods for which we have calculated absolute typed stranger is listed. The values for which it is known that it is the highest possible value are given in boldface.

num groups		1	2	3	4	5	6	7	8	9	10	11	12
subj. per type													
	2	1	3	5	7	9	11	13	15	17	19	21	23
	1 1	1	2	3	4	5	6	7	8	9	10	11	12
	3	1	1	4	4	7	7	7	7				
	1 2	1	1	3	4	5	6	7	8				
	1 1 1	1	1	3	4	5	5	7	6				
	4	1	1	1	5	5	5						
	1 3	1	1	1	4	5	5						
	2 2	1	1	1	4	5	6						
	1 1 2	1	1	1	4	5	6						
	1 1 1 1	1	1	1	4	5	5						
	5	1	1	1	1	6							
	1 4	1	1	1	1	5							
	2 3	1	1	1	1	5							
	1 1 3	1	1	1	1	5							
	1 2 2	1	1	1	1	5							
	1 1 1 2	1	1	1	1	5							
1	1 1 1 1 1	1	1	1	1	5							

x    >=x  
x    =x

  quick calculation  
  available on file  
  Avail. on file (prov.)

### **Matching: Transform**

Allows making transformations on the matching. There are three options:

**Add:** You can add a fixed number to the group ID.

**Multiply:** You can multiply the group ID with a fixed number.

**Shuffle:** You can shuffle the group IDs within the period.

With this command one can for instance easily define matching groups. Using matching groups is an important experimental technique because in a stranger matching there is generally only one independent observation per session. With matching groups one can define a stranger matching with more than one independent observation. For that, the subjects are divided into two (or more) sub-populations. Subjects are only matched within these sub-populations. Therefore these sub-populations constitute independent observations.

Example: You have 24 subjects with groups of 3 and you want to make two matching groups of 12 subjects each. You have to do the following:

1. Set the number of groups to 4.
2. Select the first 12 subjects in the parameter table and choose "Matching Stranger".
3. Select the last 12 subjects in the parameter table and choose "Matching Stranger" again.
4. Choose (the last 12 subjects are still selected) "Matching Transform". In the dialog that appears select the radio button "add" and enter 4 into the value field.
5. Set the number of groups to 8.

### ***Utilities: Make Stage Names Unique***

Changes the names of the stages by adding numbers in such a way that the names of the stages are unique. This is important because the state of the clients is defined by the stage name.

### ***Language***

Sets the language for elements that are newly created as header boxes etc. All texts within these elements can be modified later, i.e. they can also be translated later.

## **7.2.4 Questionnaire Menu**

### ***Info...***

Opens a dialog in which the parameters of the selected element can be viewed and changed.

### ***New Address Form***

The address prompt asks for the subject's address. The fields "name", "first name" and "button" are compulsory if the address form should appear. The other fields are optional. If the question texts in the

optional fields are left empty, the corresponding question do not appear at the clients' screens. If the fields “name” and “first name” are left empty, no form appears. The payment file is written whenever the last subject has passed an address form – also an empty address form

### ***New Question Form***

Question forms consist of a list of questions. The layout of the question is adjusted with rulers. All question forms except the last must contain a button.

**Name (ID):** Name of the form. The name must be unique. within a questionnaire

**Title:** Title as it appears at the clients' screens.

**Program:** A program that is conducted in the session database for the subject who enters this questionnaire. If the variable participate is set to zero, the questionnaire can be omitted.

### ***New Ruler***

Rulers are used to set the regions where labels and questions are positioned.

**Distance to left margin:** The position can be given in points or in percentage of the screen width.

**Distance to the right margin:** The position can be given in points or in percentage of the screen width.

**Distance between label and item:** This is the horizontal distance between the label and the question. Half of this distance is also kept to the left and right margins, which are defined by the options defined above.

**Size of label:** This is the maximum width of the label. When the display is given in percentage, this refers to the width between left and right margin and not to the width of the whole screen.

### ***New Question***

Questions in questionnaires correspond to items in treatments. In items, display options are defined in the field layout. Here they can be set with radio buttons. The fields are as follows:

**Label:** Name of the question as it is shown to the subjects. If no variable is set, the label appears as text over the whole width of the screen.

**Variable:** If it is an input variable, this is the name of the question as it should appear in the data file. If input is not set, it may only be one of the variables described in the chapter “Profit Display”.

**Type:** Layout of the question.

*Text, number:* A text field appears into which input is entered. In the case of the number option there is a check to see if it is really a number that has been entered. Furthermore, the check determines whether the number is in the valid range. If the option “Wide” is selected for a text field, the entry field may consist of several lines. The number of lines can be selected.

*Buttons:* A button is created for every line in the field Options. Of course, only one question with this option is permitted per question form.

*Radio buttons:* A radio button is created for every line in the field options. These buttons are arranged vertically one on top of the other.

*Slider, Scrollbar:* The slider and scrollbar make a quasi continuous entry possible. Minimum, maximum, and resolution need to be entered. In the wide layout, a label can be placed at the margins. These labels are entered in the field options. The maximum is always the value at the right border, the minimum is the value at the left border. Maximum may therefore be smaller than minimum.

*Radioline:* A radio button appears for all possible answers determined by minimum, maximum and resolution. In the wide layout, a label can be placed at the margins. These labels are entered in the field Options. The middle button can be separated from the others by entering a number greater than zero for "distance of central button". The maximum is always the value at the right most button, the minimum is the value at the left most button. Maximum may therefore be smaller than minimum.

*Radiolinelabel:* Label line for radio button lines. Labels can be placed above the buttons at the margins. The texts for this action are in the options field.

*Checkbox:* A checkbox is created for every line in the options field. These checkboxes are arranged vertically one on top of the other. The resulting value is a list of all options checked.

**Wide:** In the wide layout, the entry region for the question appears under the label over the whole width and does not only appear to the right of the label in a second column.

**Input:** This field determines whether a question is presented to the subjects (input is set) or whether a variable is only shown (input is not set).

**Empty allowed:** This is set when the question does not need to be answered.

**Minimum, maximum, resolution:** When numbers have to be entered, these values are used for checking. With sliders, scrollbars and radio lines these values are used for converting the position into a number.

**Num. rows:** In the wide layout for text entries, the number of rows that can be entered.

**Options:** The labels of buttons, radio buttons and, in the wide layout, of sliders, scrollbars and radio lines.

Example of normal layout (not wide):

This image shows a form interface in a 'normal layout' (not wide). The components are arranged in a compact, single-column fashion. At the top is a 'Text or number' input field. Below it are 'radiobuttons' labeled A, B, and C. Next are 'radiolinelabel' (left) and 'radioline' (right) with a row of six radio buttons. Below these are 'checkboxes' labeled M, N, O, and P. Then is a 'slider' with a horizontal bar and a central knob. Below the slider is a 'scrollbar' with a track and a central knob. At the bottom are three 'buttons' labeled X, Y, and Z stacked vertically.

Example of wide layout:

This image shows the same form interface in a 'wide layout'. The components are spread out horizontally to utilize more space. The 'Text or number' input field is wider. The 'radiobuttons' are still in a single column. The 'radiolinelabel' (left) and 'radioline' (right) are now labeled 'Left' and 'Right' respectively, with the row of six radio buttons positioned between them. The 'checkboxes' (M, N, O, P) are in a single column. The 'slider' is labeled 'L' and 'R' at its ends. The 'scrollbar' is also labeled 'L' and 'R' at its ends. The 'buttons' (X, Y, Z) are still stacked vertically at the bottom.

### ***New Button***

Buttons conclude question forms. You cannot define more than one button per form. However, this button can be positioned anywhere on the form. The label of a button can be selected at random. All but the last form of a questionnaire must have a button. The last form may not have a button.

### ***Check***

Checks the questionnaire.

## ***Language***

Sets the language for elements that are newly created as address forms. All texts within these elements can be modified later, i.e. they can also be translated later.

### **7.2.5 Run Menu**

#### ***Clients' Table***

Opens the clients' table. The clients' table shows which clients are connected to the server. It also contains information on the state of the clients, i.e., which screens are currently being displayed to the clients. You may open the clients' table with the first command from the Run Menu.

Each line corresponds to a subject. This subject is seated at a PC on which the client is running (column "Clients"). As long as no treatment is started, the clients can be moved. As soon as you start a treatment, the order of the clients is fixed. After a treatment has started, only those clients that are not taking part in the current session can be moved. These clients are listed at the end of the table. So long as no treatment has been started, the clients can be sorted and shuffled with the commands "Sort Clients" and "Shuffle Clients" respectively. The clients' table has five columns:

**Client:** Name of the client. In general this is the name of the subject PC. If a client is no longer connected, its name appears in brackets. If a client with the same name should reconnect, it can continue at the same place where the previous subject left off. The caption to the clients' table gives the number of clients currently connected.

**State:** The state in which the subject is at the time. Example: "Ready" when the subject is waiting for the next treatment or for the next questionnaire. The state is "\*\*\* abc \*\*\*" if the subject is in the active screen of stage abc and "- abc -" if the subject is in the waiting screen of stage abc.

**Profit:** Accumulated income made in the previous treatments. The profit made in the current treatment is *not* included therein. This profit is indicated in monetary units (Fr., €, \$, £), as various treatments may have different exchange rates.

**Show up fee invested:** TRUE if the subject has suffered losses and has declared himself ready to account for these with his/her show-up fee.

**Money added:** Extra money that the subject has injected into the game after losses in order to be able to continue. This sum is of importance for the calculation of losses. The profit in the payment file is always the net income the subject has gained from the experiment. At the hour of payment, injected money must therefore be added to the amount indicated in the payment file.

### ***Shuffle Clients***

Shuffles the clients at random.

### ***Sort Clients***

Sorts the clients. Should names have the same beginning and a number at the end, they are sorted according to the value of the number at the end, i.e., b2 comes before b11.

### ***Save Client Order***

Saves the order of the clients and how the clients can be reached over the network in order to restart them after a crash of z-Tree. When the first treatment is started, this is done automatically. The command is very useful if you have a non-arbitrary assignment of types to clients. It is a good practice to execute the command after the clients have been connected.

### ***Start Treatment***

Starts the treatment that is in the front window as soon as possible. If it is the first treatment, it is first checked whether the correct number of clients is connected. If not, a warning appears and the start can be canceled. If the treatment is allowed to start, z-Tree waits until at least as many clients are connected with the server as are needed to run the treatment. Then the treatment is started. If it is not the first treatment, the treatment starts immediately.

### ***Start Questionnaire***

Starts the questionnaire, if this is possible. A treatment needs to have been started beforehand and the clients need to be in the state Ready.

### ***xyz Table***

Opens the window showing the xyz table or brings it into the foreground.

### ***Stop Clock***

Stops the clock. However, subjects can still make entries. If, for instance, all subjects click “continue”, the session continues regardless of the stopped clock.

### ***Restart Clock***

The clock is restarted.

### ***Discard a client***

The client selected is discarded. It can be replaced by a new client. In case of a defect in a subject PC, the following needs to be done: The client in question is clicked in the clients’ window. Then the menu **Discard a client** is chosen. With this the client is discarded. The new PC is started. The client then appears in the lowermost row of the clients’ window. It first has to be selected and then moved over the old client. In doing this the mouse has to be released for an instant. If the mouse is not released this procedure only selects the space between the new and the old client.

### ***Leave Stage***

The subject, whose state is selected, leaves the stage: If a client is in an active screen, it can be moved on to the waiting screen of the stage. This option is useful when testing program. In a ordinary session is should not be necessary to use this command.

Remark: There is the variable `LeaveStage` which does the same from within a program. When this variable is set to 1, the client goes the waiting state of the current stage.

### ***Stop after this period***

When this stage will be finished, the treatment is stopped. This command is particularly useful when testing a treatment.

### ***Restart all clients***

All clients are restarted. For this z-Tree has to know the clients' addresses and the z-Leaves must be accessible over the internet (for instance, not behind a firewall). The addresses of the leaves are stored when the first treatment starts and when the command **Save Client Order** is applied.



### ***Restore Client Order***

Restore the clients' order as is has been within the last session or after the last execution of the command **Save Client Order**.

### ***Reload Database***

Reloads the database, i.e., it reloads all tables. This command is used after a crashed session.

## **7.2.6 Tools Menu**

### ***Separate Tables***

Reads a \*.xls file that z-Tree generated and creates new files: One file per table and treatment. You can select more than one file at once.

Join \*.sbj File.

Remark: The output files will override existing ones. Because the names generated are based on the original file names that should not cause problems. You cannot read in the file of the running session. Close z-Tree before separating the tables. I recommend not using the tools while a session is running.

### ***Join \*.sbj File***

Joins the \*.sbj file to a file that contains a table with a column named "Subject" (input table). Joining means that the information in the \*.sbj file, i.e., the questionnaire data, is added to each row of the input table. Which line of the \*.sbj file is entered depends on the value in the "Subject" column.

Note that the \*.sbj is transposed with respect to the \*.xls file.

**Remark:** The output file will override an existing one. Because the names generated are based on the original file names that should not cause problems. You cannot read in the file of the running session. Close z-Tree before separating the tables. I recommend not using the tools while a session is running.

### ***Join files...***

It allows to add the lines of one table to the lines of another table based on key variables. For instance, you can add the data in the subjects table (key table) to the data of the contracts table based on the key (SessionID,TreatmentNumber,Period,Subject).

**Data File:** Path of the data file. If you click the label Data File, a standard file-open dialog opens. To each row of the data file, some row of the key file will be added.

**Key File:** Path of the key file. If you click the label Key File, a standard file-open dialog opens. The key file contains the information to be added to the data file

**Link variable(s):** comma-separated list of variables in the data file.

**Key variable(s):** comma-separated list of variables in the key file.

**Outputfile:** Path of the output file. If you click the label output file, a standard save-file dialog opens. The data file contains one row per row of the data file. Each row contains the corresponding row of the data file followed by a row of the key file. The row that is taken from the key file is the row in which the key variable has the same entry as the link variable in the data file.

Example:

Data file

<b>a</b>	<b>b</b>	<b>c</b>
4	3	7
5	1	8
6	1	9

key file

<b>k</b>	<b>d</b>
1	11
2	22
3	33

data variable: b

Key variable: k

Output file

<b>a</b>	<b>b</b>	<b>c</b>	<b>k</b>	<b>d</b>
4	3	7	3	33
5	1	8	1	11
6	1	9	1	11

## ***Merge Files***

Several files containing tab-separated tables with a header (called input tables) can be merged: The output table is a table containing all the records of all the input tables. If a column does not exist in some of the input tables, the value is left empty in the rows of these tables.

Remark: The output file will override an existing one. Because the names generated are based on the original file names that should not cause problems. The tools cannot be used for modifying the files of the running session. I recommend not to use the tools while a session is running.

### ***Fix File...***

There is a severe problem in z-Tree. In some sessions the data is not correctly logged in the \*.xls file. What happens? In tables, which do not have a fixed number of records (the contracts table or user defined tables), it can happen that records are stored more than once. This occurs only if there are fewer records in a later period than in an earlier period. In this case it occasionally happens that the table is "filled" with old data.

Fortunately, this error can easily be recognized in the file. The command "Fix File..." corrects a corrupt file. It also gives you a message whether there was actually an error in the file. Please apply this command to the \*.xls files of sessions where you worked with contract table or with user defined tables that did not have a predefined number of records.

Remarks: A test for this error is also included in the command **Separate tables**. There is no problem if you use only the globals and the subjects table.

## **7.2.7 Windows Menu**

### ***Treatment and questionnaire files***

All treatments and questionnaires opened in z-Tree can be brought into the foreground in the windows menu.

### ***Toolbar***

Opens and closes the toolbar.

### ***Status Bar***

Opens and closes the status bar.

## **7.2.8 Help Menu**

(unimplemented help function)

## 7.3 Programming environment

### 7.3.1 Programming language

---

#### Error handling

There is no exception handling in z-Tree. You have to care the parameters of a function and the arguments of operators are valid. Illegal values can cause z-Tree to crash.

---

#### Constants

TRUE

FALSE

---

#### Operators

Mathematical operators

+      addition

-      subtraction

\*      multiplication

/      division

Relational operators:

<      smaller

<=    smaller or equal

==    equals

!=    unequal

>=    greater or equal

>      greater

Logical operators:

&      logical and

|      logical or

Scope operators:

:      next higher scope

\ highest possible scope. This is always the record of the globals table.

---

## Statements

In the following, a and b are logical expressions, x, y, and z numeric expressions, v is the name of a variable, t is the name of a table, s is one statement, ss1 and ss2 are lists of statements.

<code>x = y;</code>	Assigns the expression y to the variable x.
<code>if (a) {ss1}</code>	If a returns TRUE, the statements ss1 are executed
<code>if (a) {ss1} else {ss2}</code>	If a returns TRUE, the statements ss1 are executed; otherwise the statements ss2 are executed.
<code>if (a) {ss0} elsif (b1) {ss1} elsif (b2) {ss2}....</code>	If a returns TRUE, the statements ss0 are executed; otherwise, if b1 returns TRUE, the statements ss1 are executed and so on.
<code>if (a) {ss0} elsif (b1) {ss1} elsif (b2) {ss2}.... else {sse}</code>	If a returns TRUE, the statements ss0 are executed; otherwise, if b1 returns TRUE, the statements ss1 are executed and so on. If none of either the a or bx-statements are TRUE , the statements sse are executed.
<code>t.do{ ss }</code>	in the table t, the statements ss1 are executed for all records
<code>t.new{ ss }</code>	in the table t, a new record is generated and in this record the statements ss1 are executed
<code>array v[ x ];</code>	defines an array, i.e., an indexed variable. Array variable can be accessed with <code>v[ x ]</code> . Index runs from 1 to x in steps of 1.
<code>array v[ x, y ];</code>	defines an array, i.e., an indexed variable. Array variable can be accessed with <code>v[ x ]</code> . Index runs from x to y in steps of 1.
<code>array v[ x, y, z ];</code>	defines an array, i.e., an indexed variable. Array variable can be accessed with <code>v[ x ]</code> . Index runs from x to y in steps of z.
<code>while( a ) { ss }</code>	While a is TRUE, the statements ss are executed. (Loops can be left with the key combination <Ctrl>-<Alt>-[F5].)

`repeat { ss } while ( a );` the statements `ss` are executed. Then it is checked whether `a` is TRUE as long as `a` is TRUE `ss` is repeated. (Loops can be left with the key combination <Ctrl>-<Alt>-[F5].)

`later ( a ) do { ss }` The expression `a` is calculated. The resulting seconds later the statements `ss` are executed. The statements `ss` are executed in the exact same scope environment in which this `later` command was called. If `a` is negative, `ss` is not called.

`later ( a ) repeat { ss }` The expression `a` is calculated. The resulting seconds later the statements `ss` are executed. The statements `ss` are executed in the exact same scope environment in which this `later` command was called. After this, condition `a` is calculated again and `a` seconds later `ss` is executed again. If `a` is negative the `later` command is cancelled. When the stage is left in which this `later` command was started, is also cancelled.

---

## Functions

In the following, `a` and `b` are logical expressions, `x` and `y` numeric expressions.

<code>abs( x )</code>	Absolute value of <code>x</code> .
<code>and( a,b )</code>	TRUE if and only if <code>a</code> and <code>b</code> are true.
<code>atan( x )</code>	Arctangent of <code>x</code> .
<code>cos( x )</code>	Cosine function of <code>x</code> .
<code>exp( x )</code>	Exponential of <code>x</code> ; $e^x$
<code>gettime( )</code>	The number of seconds since the computer was started.
<code>if( a,x,y )</code>	If <code>a</code> , then the value of the function is <code>x</code> , otherwise <code>y</code> .
<code>ln( x )</code>	Natural logarithm of <code>x</code> .
<code>log( x )</code>	Base -10 logarithm of <code>x</code> .
<code>max( x,y )</code>	Maximum of <code>x</code> and <code>y</code> .
<code>min( x,y )</code>	Minimum of <code>x</code> and <code>y</code> .
<code>mod( x,y )</code>	Remainder after <code>x</code> is divided by <code>y</code> .
<code>not( a )</code>	TRUE if and only if <code>a</code> is <i>not</i> true.
<code>or( a,b )</code>	TRUE if and only if <code>a</code> or <code>b</code> is true.

<code>pi()</code>	3.1415...
<code>power( x,y )</code>	$x^y$ , if x is positive. If x is negative and y is an odd number or 1/y is an odd number, $x^y$ is returned, otherwise $ x ^y$ is returned.
<code>random()</code>	Uniformly distributed random number between 0 and 1.
<code>randomgauss()</code>	Normally distributed random number with average 0 and standard deviation 1.
<code>randompoisson( x )</code>	Poisson distributed random number with average x (the result is a whole number).
<code>round( x,y )</code>	Rounds x to a multiple of y, i.e., supplies the multiple of y that is closest to x. Examples: <code>round( 345.67, 20 ) == 340</code> , <code>round( 345.67, 0.1 ) == 345.7</code> , <code>round( 2.5, 1 ) == 3</code> , <code>round( -2.5, 1 ) == -3</code> .
<code>rounddown( x,y )</code>	Rounds x down to a multiple of y, i.e., it returns the greatest multiple of y that is smaller or equal to x. This definition is also employed for negative numbers. Example: <code>rounddown( -3.6,1 ) == -4</code> .
<code>roundup( x,y )</code>	Rounds x up to a multiple of y, i.e., it returns the smallest multiple of y that is greater than or equal to x. This definition is also employed for negative numbers. Example: <code>roundup( -3.6,1 ) == -3</code> .
<code>same( x )</code>	Short form for <code>x == x</code> Therefore, this function is only feasible in table functions. x may also be an expression.
<code>sin( x )</code>	Sine of x.
<code>sqrt( x )</code>	Square root of x.

---

### Table functions

<code>average(x)</code> , <code>average( a,x)</code>	Average of the (found) values.
<code>count()</code> , <code>count(a)</code>	Number of records in the table or number of found records.
<code>find(x)</code> , <code>find(a,x)</code>	The first value of x (where a is satisfied).
<code>maximum(x)</code> , <code>maximum( a,x)</code>	Maximum of the (found) values.
<code>median(x)</code> , <code>median( a,x)</code>	Median of the (found) values.
<code>minimum(x)</code> , <code>minimum( a,x)</code>	Minimum of the (found) values.

<code>product(x), product( a,x)</code>	Product of the (found) values.
<code>regressionslope( x,y)</code>	Gradient of a linear regression through the (found) points (x, y).
<code>regressionslope( a,x,y)</code>	
<code>stddev(x), stddev( a,x)</code>	Standard deviation of the (found) values.
<code>sum(x), sum(a,x)</code>	Sum of the (found) values.

---

## Iterator

<code>iterator(v)</code>	creates a small table with the variable v. (see
<code>iterator(v, x)</code>	the chapter on iterator)
<code>iterator(v, x, y)</code>	
<code>iterator(v, x, y, z)</code>	

---

## Syntax diagram of the programming language

The following syntax description is intended to serve as a reference to experts.

### Syntax used to express the grammar

<code>a : b;</code>	: a can be replaced by b
<code>a b</code>	: a followed by b
<code>a   b</code>	: a or b
<code>"x"</code>	: exactly that text
<code>a*</code>	: zero, one or more repetitions of a
<code>(a)</code>	: a
<code>'x'</code>	: the character x

### Definitions

```

digit19 : '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9';
digit09 : '0' | digit19;

letter : '_' |
        'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' | 'l' | 'm' |
        'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v' | 'w' | 'x' | 'y' | 'z' |
        'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' | 'H' | 'I' | 'J' | 'K' | 'L' | 'M' |
        'N' | 'O' | 'P' | 'Q' | 'R' | 'S' | 'T' | 'U' | 'V' | 'W' | 'X' | 'Y' | 'Z';

integer : ( digit19 digit09* ) | '0';
realdigit : integer '.' digit09 * | '.' digit09 digit09* ;
real : realdigits ( ('e' ('+' | '-' | "" ) integer) | "");

```

### Terminal elements

NAME	: letter (letter digit)*;
NUMBER	: real;
BOOLVALUE	: "TRUE"   "FALSE";
FUNCTIONNAME0	: "gettime"   "pi"   "random"   "randomgauss" ;



```

FUNCTIONNAME1      : "abs" | "atan" | "cos" | "exp" | "ln" | "log" | "sin" | "sqrt" |
                    "randompoisson";
FUNCTIONNAME2      : "max" | "min" | "mod" | "power" | "round" | "rounddown" |
                    "roundup";
BOOLFUNCTIONNAME1  : "not";
BOOLFUNCTIONNAME2  : "and" | "or";
BOOLFUNCTIONNAME1R : "same" ;
TABLEFUNCTION0     : "count";
TABLEFUNCTION1     : "sum" | "product" | "average" | "stddev" | "minimum" | "maximum"
                    | "median" | "find" ;
TABLEFUNCTION2     : "regressionslope" ;
BOOLOPERATOR2      : "&" | "|";
COMPOPERATOR       : "<" | "<=" | "==" | "!=" | ">=" | ">" ;
SCOPEOPERATOR      : ":" | "\";

```

## Grammar

```

program            : statementlist;

statementlist      :
                    | statement statementlist
                    ;

statement          : ';'
                    | lvalue '=' expression ';'
                    | "while" '(' boolexpression ')' '{' statementlist '}'
                    | "repeat" '{' statementlist '}' "while" '(' boolexpression ')' ';'
                    | "later" '(' expression ')' "do" '{' statementlist '}'
                    | "later" '(' expression ')' "repeat" '{' statementlist '}'
                    | "do" '{' statementlist '}'
                    | tableidentifier '.' "do" '{' statementlist '}'
                    | "new" '{' statementlist '}'
                    | tableidentifier '.' "new" '{' statementlist '}'
                    | "if" '(' boolexpression ')' '{' statementlist '}'   elseclause
                    | "array" arraydeclaration ';'
                    ;

elseclause         :
                    | "elsif" '(' boolexpression ')' '{' statementlist '}'
                    elseclause
                    | "else" '{' statementlist '}'
                    ;

```

```

lvalue          : variable
                  ;

arraydeclaration : IDENTIFIER '[' ']'
                  | IDENTIFIER '[' expression ']'
                  | IDENTIFIER '[' expression ',' expression ']'
                  | IDENTIFIER '[' expression ',' expression ',' expression ']'
                  ;

expression       : summand
                  | '+' expression
                  | '-' expression
                  | expression '+' expression
                  | expression '-' expression
                  ;

summand          : factor
                  | summand '*' summand
                  | summand '/' summand
                  ;

factor           : NUMBER
                  | variable
                  | '(' expression ')'
                  | FUNCTIONNAME0 '(' ' ')'
                  | FUNCTIONNAME1 '(' expression ')'
                  | FUNCTIONNAME2 '(' expression ',' expression ')'
                  | "if" '(' boolexpression ',' expression ',' expression ')'
                  | tablefunction
                  ;

variable :       IDENTIFIER
                  | arrayvariable
                  | SCOPEOPERATOR variable
                  ;

arrayvariable :  IDENTIFIER '[' expression ']'

tablefunction :  puretablefunction
                  | tableidentifier '.' puretablefunction
                  ;

```

```

puretablefunction : TABLEFUNCTION0 '(' ')'
                  | TABLEFUNCTION0 '(' boolexpression ')'
                  | TABLEFUNCTION1 '(' expression ')'
                  | TABLEFUNCTION1 '(' boolexpression ',' expression ')'
                  | TABLEFUNCTION2 '(' expression ',' expression ')'
                  | TABLEFUNCTION2 '(' boolexpression ',' expression ',' expression
                    ')'
                  ;

tableidentifier : IDENTIFIER
                | ITERATOR '(' IDENTIFIER ')'
                | ITERATOR '(' IDENTIFIER ',' expression ')'
                | ITERATOR '(' IDENTIFIER ',' expression ',' expression ')'
                | ITERATOR '(' IDENTIFIER ',' expression ',' expression ','
                  expression ')'
                ;

boolexpression   : boolatom
                  | boolatom OPBOOL boolexpression
                  ;

boolatom        : BOOLFUNCTIONNAME0
                  | expression OPCMP expression
                  | '(' boolexpression ')'
                  | BOOLFUNCTIONNAME1 '(' boolexpression ')'
                  | BOOLFUNCTIONNAME2 '(' boolexpression ',' boolexpression ')'
                  | BOOLFUNCTIONNAME1R '(' expression ')'
                  ;

```

### 7.3.2 Scope environment

When a program is executed, it is always executed within a particular scope environment. In the following, we give a complete description in what environment the programs are called.

table	Program lies in	Owner var.	Cond.	Records for which the program is run	Records accessible through scope operator
globals			without	Record of globals table	-
			with	If cond. is met, record of globals table	-
subjects			without	All records of subjects table	globals
			with	Records of subjects table for which cond. is met	globals
summary			without	Record of current period	globals
			with	Record of current period, if cond. is met	globals
contracts	stage	without	without	All records of contracts table	globals
		without	with	All records of contracts table that meet cond.	globals
		with	without	All records of contracts table. The records are worked through subject for subject. The records of a subject are those records for which the Owner variable equals the Subjects variable.	Record of owner in subjects table, globals
		with	with	Same as above, except that the record of cond. also needs to be met	Record of owner in subjects table, globals
contracts or user defined tables	standard box, grid box, contract grid box		without	All records of contracts table	Record in subjects table of subject who has clicked the button globals
			with	All records of contracts table that meet cond.	ditto
contracts or user defined tables	contract creation box, contract list box		without	All new records of contracts table in the case of the contract creation box, and the selected record in the case of the contract list box. (To run the program for all records cond. has to be set to TRUE)	ditto
			with	All records of contracts table that meet cond.	ditto

### 7.3.3 Program execution

Programs at the beginning of a stage are executed one after the other. If the **Start** option in the stage is set to **Wait for all**, This proceeds as follows: First, the first program is executed for all subjects. Then the next program is executed for all subject and so forth. Note, that setting the value of **Participate** to zero does not stop program execution. **Participate not** is considered but when all programs have been executed.

If another **Start** option is set, first, the first program is executed for the subject who entered first. Then the second program is executed for this subject and so for. When all programs for the first subject are executed, then the programs for the next subject are execute. In this case not all programs are executed for all subjects. The programs in the **globals** table are only executed for the first subject, programs for the **summary** table only for the last one. If there is an owner variable in a program of the **contracts** table (or a user defined table), the program is only executed for the records of the owner. This means that is executed for the records in which the owner variable equals the value of the **Subject** variable of this subjects.

At the beginning of each period, the programs are executed in the following order:

1. Standard variables are set.
2. Programs in background
3. Subject program (in current period) in the subjects table (the program in the specific parameter)
4. Role program in subjects table (in the role parameter)
5. Period program in globals table (in the period parameter)
6. Programs of the first stage

### 7.3.4 The tables and their standard variables

---

#### The subjects table

Table that contains a record for each subject. This is the main table. It is freshly set up for each period. The **subjects** table of the preceding period is available under the name **OLDsubjects**. Values from earlier periods than the immediately preceding one are not accessible.

In the subjects table, the variables Period, Subject, Group, Profit, TotalProfit and Participate are always defined.

**Period:** Number of the period. The first paying period is the period 1. Thus, if there are 3 trial periods, the first period has the number -2.

**Subject:** Number of subject; starts at 1.

**Group:** Group number as it is displayed in the parameter table (possibly modified by a program).

**Profit:** Profit made in this period (in experimental currency units). Needs to be calculated; default is zero. Profit is the relevant variable for payment.

**TotalProfit:** Total profit made in this treatment. This is calculated automatically. It should not be changed.

**Participate:** Indicates whether the subject is taking part in the current stage or not. If Participate equals 1, he or she is taking part, if 0, he or she is not taking part. Default is 1.

**LeaveStage:** If set to 1, causes a subject to leave an active state and to move to waiting state.

---

### The globals table

This table contains a single record, i.e., the cells in this table are In it, values are stored that are the same for all subjects as for instance global variables. It is freshly set up for each period. The globals table of the preceding period is available under the name OLDglobals. It contain the following variables per default:

**Period:** As in subjects table.

**NumPeriods:** ID of the last period.

**RepeatTreatment:** If this variable is greater than zero after the last period, the treatment is run again with the same number of periods. The period counter is incremented as if all would be done in one treatment.

---

### The summary table

The summary table contains one record per period. This record is not destroyed when the period has been finished. The whole table is destroyed after the treatment has been finished. This table is most useful for observing aggregate data of the treatment.

**Period:** Number of period as in subjects table.

---

### **The session table**

As the subjects table, this table contains one record per subject. However, this table is not freshly set up after each period. It contains the aggregated profits earned in earlier treatments. You can also use it to exchange information between treatments.

**Subject:** Subject ID

**FinalProfit:** Income of the subject not including the show up fee.

**ShowUpFee:** show up fee.

**ShowUpFeeInvested:** 1, if subject agreed to invest show up fee to be able to proceed in the experiment.

**MoneyAdded:** Credit limit of the subject.

**MoneyToPay:** = FinalProfit + ShowUpFee + MoneyAdded

**MoneyEarned:** = FinalProfit + ShowUpFee

---

### **The contracts table**

This table is used mainly for market experiments. New records can be added to this table and existing records can be changed. The chapter "Market Experiments" explains the procedure in detail.

**Period:** Period in which record was created.

---

### **The table of the previous period "OLDtables"**

The data of past periods can be accessed in the OLD-tables. You just have to prefix the table name with OLD. Data of even earlier period have to be accessed indirectly or by using a table with lifetime treatment or session.

---

### **User defined tables**

User defined tables can to be declared at the beginning of the treatment.

**Period:** Period in which record was created.

## 7.4 Text formatting

### 7.4.1 Variable output in test display

Besides putting a variable in to an item, there is a second way to display the value of a variable. They can also be inserted in text. The syntax is

```
< variablename | layout >
```

In the output, this expression is replaced by the value of the variable in the given layout. Layout can be a layout command as it is defined in the layout field of the item. Of course, only the number and the “!text” options are valid. For instance, you cannot insert buttons in this way.

Variables can be inserted in labels and in the options of a layout of an item or in the text of a help box. So it is for instance possible that the text of a radio button depends on what happened before in the experiment. If a text contains inserted variables it must be preceded by <>, otherwise the text will not be processed. The characters <> have to appear at the very beginning of the dialog field. It is also possible to insert variables in the options of the layout of an inserted variable, i.e., to nest inserted variables. If a variable appears in an option of the same variable, the variable name can be omitted. So

```
< x |!text: -1="negative"; 1="< x | 1>">
```

is a shorter form for

```
< x |!text: -1="negative"; 1="< x | 1>"> .
```

**Warning:** Be careful when you use this option for items that can change their value. The width of an item is calculated when the screen is displayed first and it is not modified afterwards. So, if the value of a variable changes from 1 to 20, only 2 might be displayed. You are responsible to avoid it. You can choose a sufficiently wide first value or place the item into a box with other items that are wider. (You could use an invisible variable that contains a lot of spaces.)

Note further that labels are evaluated only once, at the beginning of the stage. Variables in labels are not updated.



## 7.4.2 Formatting with RTF commands known to be processed

In some places, texts can be formatted with RTF: Labels can be formatted in standard boxes, grid boxes, contract creation boxes and contract grid boxes. The options in the “!text” layout can be formatted in standard boxes, grid boxes, contract creation boxes and contract grid boxes. Finally, texts in help boxes and message boxes can be formatted.

The RTF format begins with “{\rtf “ (with a blank space at the end) and ends with “}”. In between is the text to which formatting instructions can be added. In option of “!text” layouts, the “{\rtf “ has to appear in every option:

```
!text: 1="{\rtf \b one}"; 2="{\rtf \b two}"; )
```

Formatting instructions begin with “\” and end with a blank space. If a formatting option is supposed to apply only to a certain range, you can place this range in curly brackets. Not the whole RTF is supported. The most important formatting instructions that are supported are:

\tab	tabulator
\par	new paragraph
\line	new line
\bullet	thick dot •
\ql	aligned to left
\qr	aligned to right
\qc	centered
\b	bold
\b0	not bold
\i	italic
\i0	not italic
\sub	small and inferior numbers (index)
\super	small and superior numbers (exponent)
\strike	crossed through
\ul	underline
\ul0	do not underline
\cfn	Text color. n is the index of the color table which is defined by colortbl. See example.

`\fsn` Font size in units of half a dot. The font size must be explicitly given, otherwise it is larger (24) than usual in z-Leaf.

### Examples

normal font size, <b>bold</b> , no longer bold	<code>{\rtf \fs18 normal font size, \b bold, \b0 no longer bold }</code>
------------------------------------------------	------------------------------------------------------------------------------

Text <i>italic</i> no longer italic new line	<code>{\rtf \fs18 Text {\i italic} no longer italic \par new line}</code>
-------------------------------------------------	-------------------------------------------------------------------------------

One word in <b>ochre</b> , rest in black	<code>{\rtf \colortbl;\red0\green\blue0;\red255\g reen100 \blue0;} \fs18 One word in \cf2 ochre\cf0 , the rest in black.}</code>
------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------

For more complex operations it is best to format the text in a word processor and then export it as RTF. However, if you make the RTF code by hand, it will be shorter and easier to read.

The insertion of variables is carried out *before* the interpretation of RTF. This makes conditional formatting possible as in the following example: When the variable `Bold` is 1, “hallo” should be shown in boldface but otherwise in plain text:

```
<> {\rtf <BOLD |!text: 0="";1="\b ";>hallo}
```

RTF commands known to be processed

### 7.4.3 Combining RTF and inserted variables

The insertion of variables is carried out *before* the interpretation of RTF. This makes conditional formatting possible as in the following example: When the variable `BoldVar` is 1, “hallo” should be shown in boldface but otherwise in plain text:

```
<> {\rtf <BoldVar |!text: 0="";1="\b ";>hallo}
```

## 7.5 Command line options

Command line options common for z-Tree and z-Leaf:

`/language lan` This option sets the language as the menu command "language". In z-Tree it concerns the default element of a treatment and questionnaire. In z-Leaf it is relevant for the general error messages. The default value is German. If you want to change this in your environment, you can create a shortcut for z-Tree that initializes the language to your choice.

In the following list, we give all the options available. For instance, to use Finnish, you can use the option `/language suomi`. Note that for the Russian and for the Ukraine version, you need a Russian or Ukraine operating system.

Brasil	br, brasil, brasil
Catalan	cat, catala, catalan
Dutch	nl, nederlands, dutch
English	en, english, english
Finnish	fi, suomi, finnish
French	fr, francais, french
German	de, deutsch, german
Italian	it, italiano, italian
Norwegian (bokmål)	no, norsk, norwegian
Norwegian (nynorsk)	nyno, nynorsk, newnorwegian
Polish	pl, polski, polish
Portugues	pt, portuguese, portugues
Russian	ru, ruskii, russian
Spanish	es, espanol, spanish
Swedish	se, svensks, swedish
Swiss German Zurich style	zh, zuerituetsch, zurichgerman
Turkish	tr, turkce, turkish
Ukraine	ua, ukrainska, ukraine

`/channel ch` Determines the channel through which the z-Leaves establish contact to z-Tree. The actual channel is 700+ch. If you want to run more than one z-Tree on one computer they must work with different channels, i.e., you set the channel to values greater than one.

Command line options common for z-Tree:

`/xlsdir dir` Sets the directory where the xls file is stored.  
`/sbjdir dir` Sets the directory where the subjects file is stored.  
`/datadir dir` Sets the directory where the xls and the subjectsfile are stored.  
`/adrdir dir` Sets the directory where the address file is stored.  
`/paydir dir` Sets the directory where the payment file is stored.

`/privdir dir`     Sets the directory where the adress and the payment file are stored.  
`/gsfdir dir`     Sets the directory where the GameSafe is stored.  
`/tempdir dir`     Sets the directory where the temporary files `@lastclt.txt`, `@db.txt` and `@prevdb.txt` are stored.  
`/datadir dir`     Sets the directory where the treatment and questionnaire files are automatically stored (before they run).  
`/leafdir dir`     Sets the directory where the file `server.eec` is stored

Command line options common for z-Tree:

`/server adr`     `adr` contains the IP address of experimenter PC.  
`/name name`     `name` will be the name the client has in the clients' table.  
`/size widthxheight`  
                     Size of the client's window for the format *widthxheight*. In this way you can test on a large screen how the layout will look like on a small screen. For example, the VGA format is 640x480. There may be no spaces around the x.  
`/position xpos,ypos`  
                     Position of the client's window with respect to the top left corner. There may be no spaces around the comma.

## 7.6 The Import Format

Treatments and questionnaires can be brought into a readable format with "Export". This format may also be read in again. This format is not described in detail here. Generally, however, the elements of the stage tree are given a form such as the following:

```

element name {
    option1 = value1;
    option2 = value2;
    subelement1
    subelement2
}

```

The sub elements have the same structure. Exporting a treatment or a questionnaire supplies the options.