

A Comprehensive Introduction to z-Tree



Stefan Palan

stefan.palan@uni-graz.at

<http://www.palan.biz/academic>

References

- Fischbacher, U., *z-Tree Tutorial*, www.iew.uzh.ch/ztree/ztree21tutorial.pdf, 3rd of January, 2002. (Tutorial)
- Fischbacher, U., *z-Tree – Zurich Toolbox for Readymade Economic Experiments - Experimenter's Manual*, Institute for Empirical Research in Economics, University of Zurich, Working Paper No. 21, ISSN 1424-0459, 1999. (Manual)
- Fischbacher, U., *z-Tree Reference Manual*, <http://www.iew.uzh.ch/ztree/ztree21ref.pdf>, 6th of January, 2006. (Reference Manual)
- Z-Tree Wiki, <https://www.uzh.ch/iew/ztree/ssl-dir/wiki/>, 26th of July, 2009. (Wiki)
- Lecture resources: www.palan.biz/academic - select Downloads

Part I

A Simple Example

Structuring a typical session 1/3

1. Subject arrival
 - Subjects checked on list
 - Excess subjects are sent home with show-up fee
 - Other subjects are (randomly) assigned to workstations
2. Instructions on mechanism
 - Hand out printed instructions, read them out loud
 - Control questions
3. Training round(s)
4. Bathroom break

Structuring a typical session 2/3

5. Instructions on parameters

- Hand out printed instructions, read them out loud
- Control questions

6. Run treatment

...Repeat 5 & 6 for multiple parameter constellations...

7. Questionnaires

- Solicit questionnaire responses
- Solicit risk-aversion, etc.

Structuring a typical session 3/3

8. Payment

- Random number generation
 - As transparent as possible (e.g. real dice)
 - Random numbers for individual subjects reduce cost variance but increase required time
- Pay subjects individually and anonymously
- Have subjects sign receipt
- Ask subjects not to talk about experiment with others

General rules

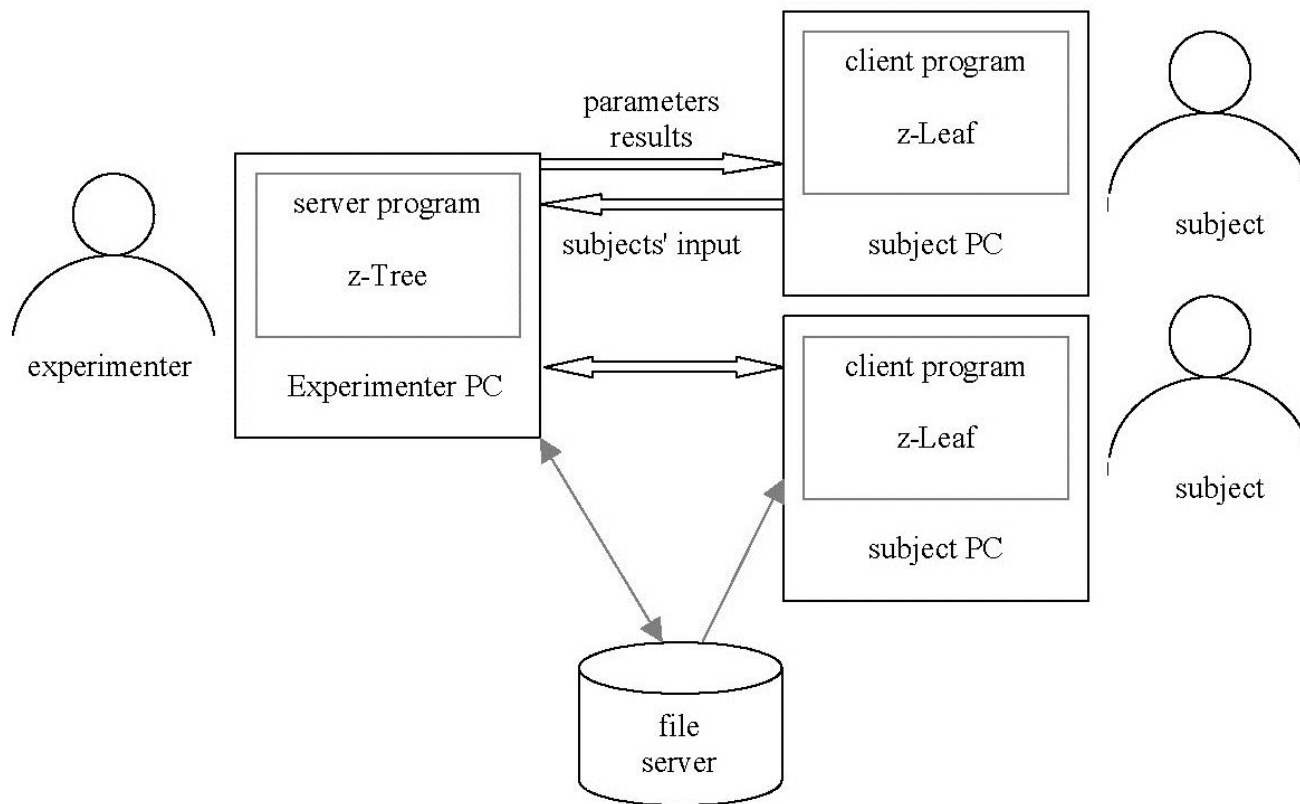
- Follow session structure cheat sheet diligently
 - Contains procedural steps
 - Contains program settings
 - Contains required material (dice, cards, etc.)
- Note special occurrences in experimenter diary
- Backup everything

Characteristics and features

Zurich Toolbox for Readymade Economic Experiments

- Simple, flexible programming language for economic experiments
- Client-server architecture
- Automatic networking, data collection and payoff calculation
- Crash recovery capability

Network topology



Source: Tutorial, p. 6.

Network connections: Critical issues

- Setup with file server
 - Ensure server has read/write access
 - Ensure client has read access
 - Put all files into one directory
 - Clients can be restarted by *Run-Restart All Clients*

- Setup without file server
 - Ensure server has read/write access
 - Point client to server IP
 - Clients can only be restarted manually

Network connections: Critical issues

- Server IP (in server.eec)
 - Command line parameter, e.g.: `zleaf /server 10.0.0.1`
 - IP file is in the `zleaf.exe`-directory
 - IP file is in the directory `c:\expecon\conf`
 - IP equals local machine's IP

- Client name
 - Command line parameter, e.g.: `zleaf /name pc1`
 - File `name.eec` in `zleaf.exe`-directory
 - Name equals local machine's TCP/IP hostname

Other important command line options

- zleaf.exe
 - Language: /language
e.g. /language en, or /language english
 - Screen resolution: /size
e.g. /size 1024x768

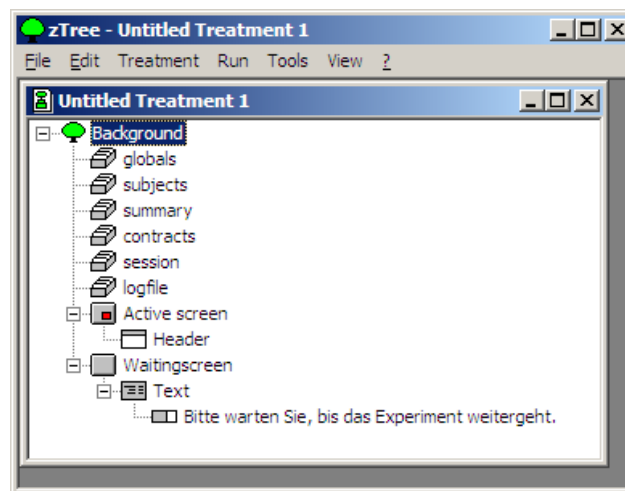
- ztree.exe
 - Change default directories
 - /xlmdir, /sbjdir, /adrrdir, /paydir, /gsfdir, /tempdir, /datadir
(autosaves of .ztt and .ztq), /leafdir (server.eec)
 - e.g. /adrrdir c:\data

Files used by z-Tree


ztree.exe	Server program file
zleaf.exe	Client program file
name.ztt	Treatment code file
name.ztq	Questionnaire code file
name.txt	Parameter input file
@1.ztt, @2.ztt, ... @1.ztq, @2.ztq, ...	Backups of treatments and questionnaires
@db.txt, @lastclt.txt, @prevdb.txt	Temporary files
090330_0804.adr	Subject address file
090330_0804.gsf	Crash recovery file (binary)
090330_0804.pay	Payout information file
090330_0804.sbj	Questionnaire response file
090330_0804.xls	Main data file (~ MS Excel compatible)

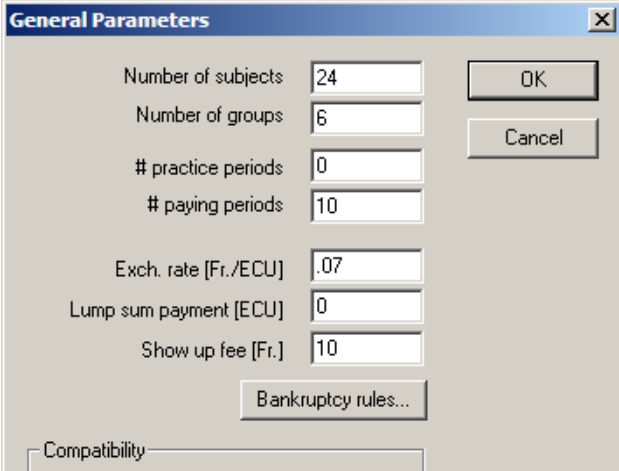
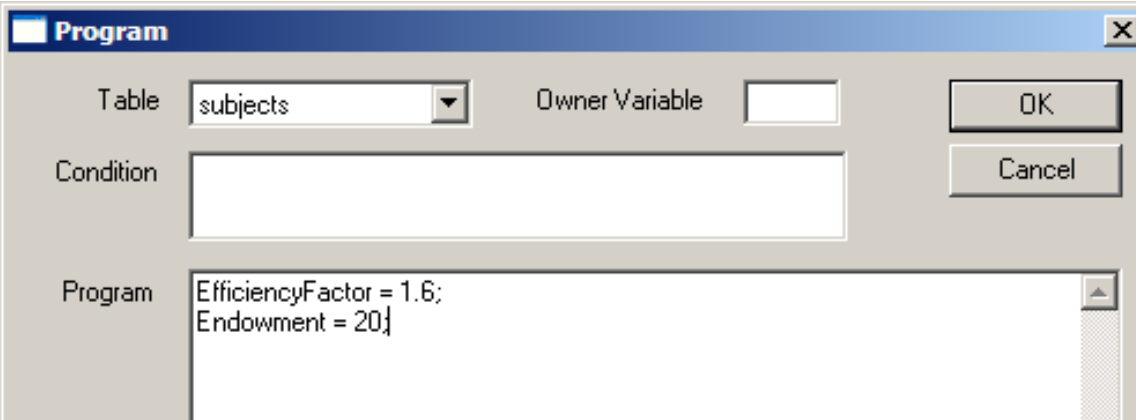
Example: Public goods experiment

- Groups of $n = 4$ subjects
- Initial endowment of subject i : $w_i = 20$
- Contribution of subject i : $c_i \in [0, w_i]$
- Profit of subject i : $\pi_i = w_i - c_i + 1.6 \cdot \frac{1}{n} \cdot \sum_{j=1}^4 c_j$
- New z-Tree treatment:



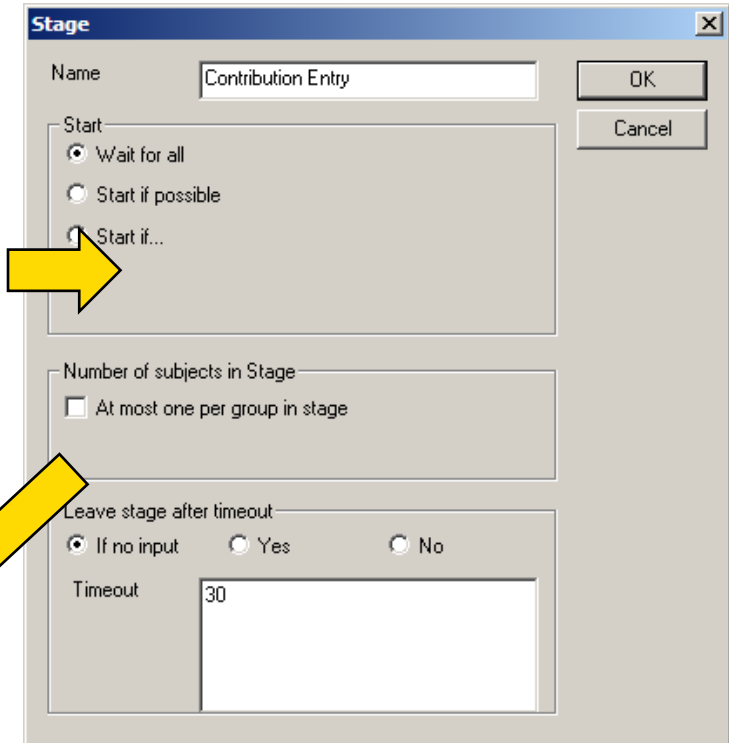
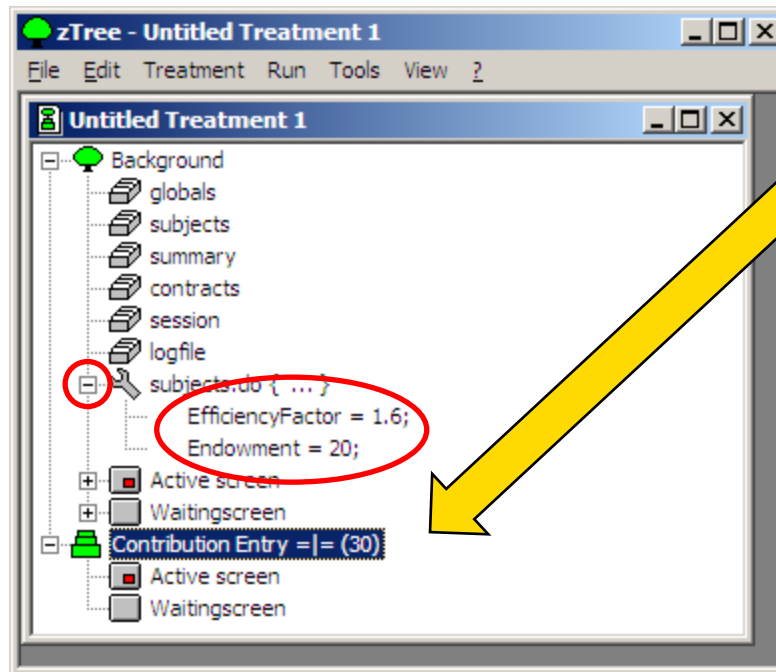
Define parameters

- Define general parameters in *Background*: 
- Create new program
 - Select *logfile* in stage tree
 - Click *Treatment-New Program...*

The first *stage*

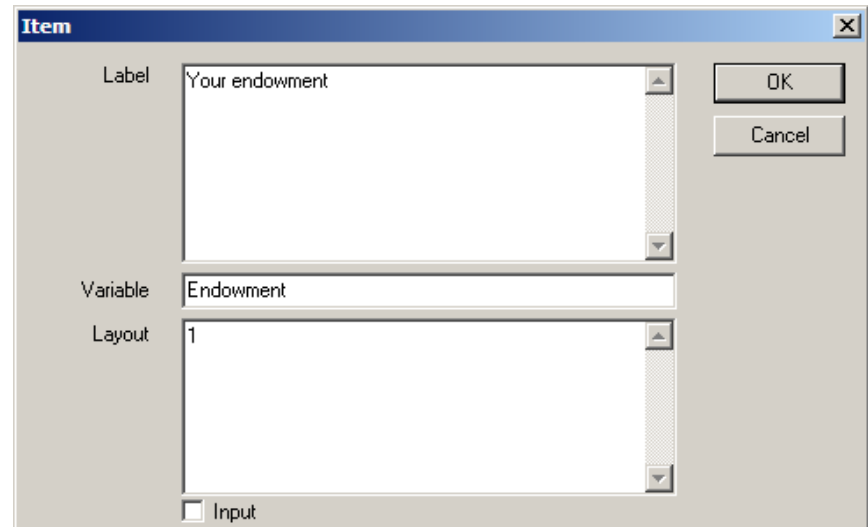
- Add the first stage
 - Select the Background
 - Click *Treatment-New Stage...*



Designing the screen 1/2

- Add the first box
 - Select *Active Screen* in the *Contribution Entry* stage
 - Click *Treatment-New Box>Standard Box...*
 - Click *OK*

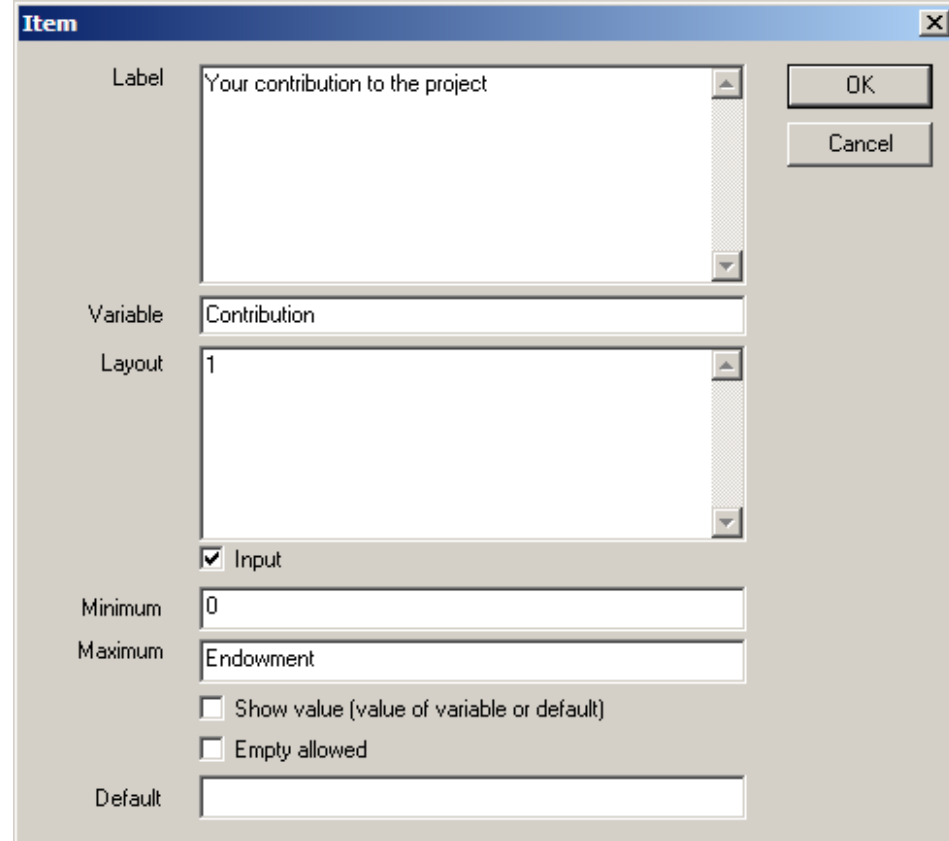
- Add an output item
 - Select the new *Standard Box*
 - Click *Treatment-New Item...*



Designing the screen 2/2

- Add an input item
 - Select the first item
 - Click *Treatment-New Item...*

- Add a button
 - Select the input item
 - Click *Treatment-New Button...*
 - Enter “OK” as name and click *OK*

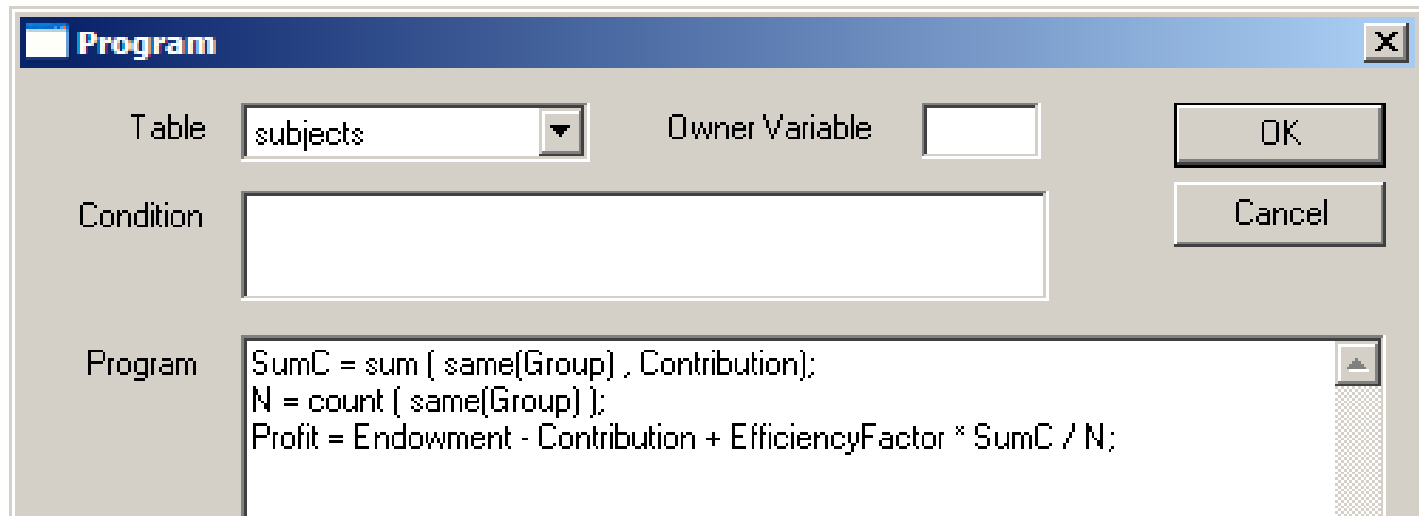


The screenshot shows the 'Item' dialog box with the following configuration:

- Label: Your contribution to the project
- Variable: Contribution
- Layout: 1
- Input:
- Minimum: 0
- Maximum: Endowment
- Show value (value of variable or default):
- Empty allowed:
- Default: (empty)

Profit calculation

- Add new stage *Profit Display*
- Add profit calculation
 - Select *Profit Display* stage
 - Click *Treatment-New Program...*



Program

Table: subjects

Owner Variable:

Condition:

Program:

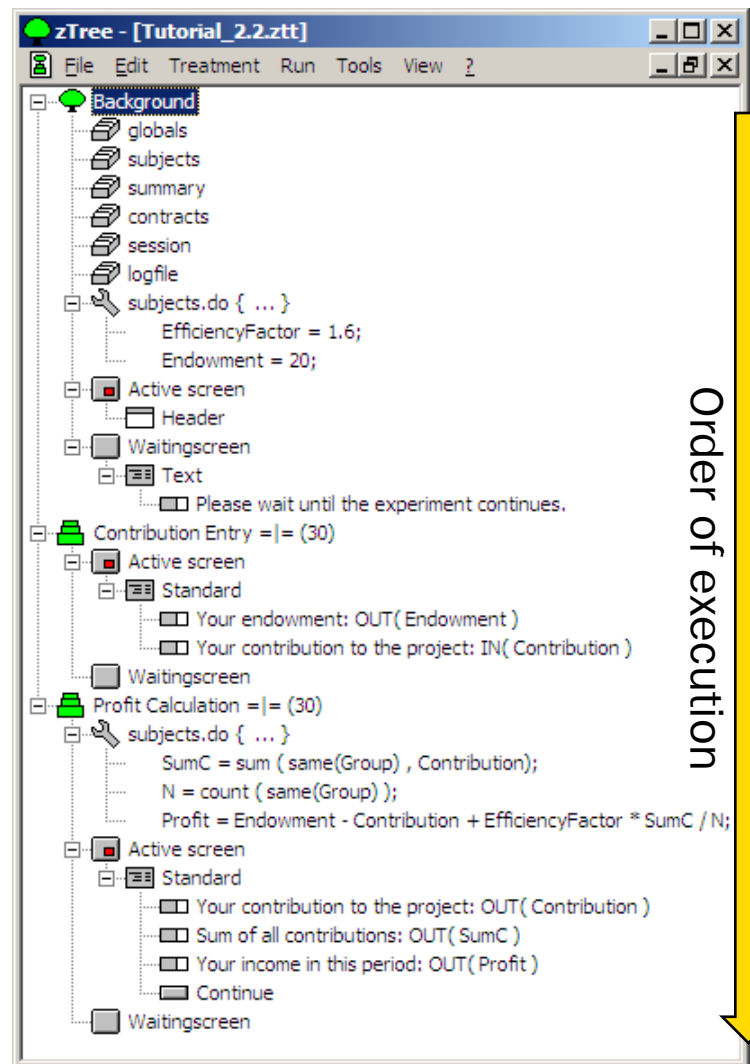
```
SumC = sum [ same(Group) , Contribution];  
N = count [ same(Group) ];  
Profit = Endowment - Contribution + EfficiencyFactor * SumC / N;
```

OK

Cancel

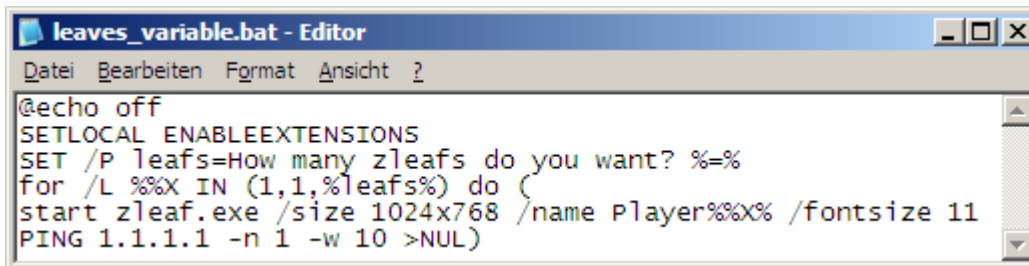
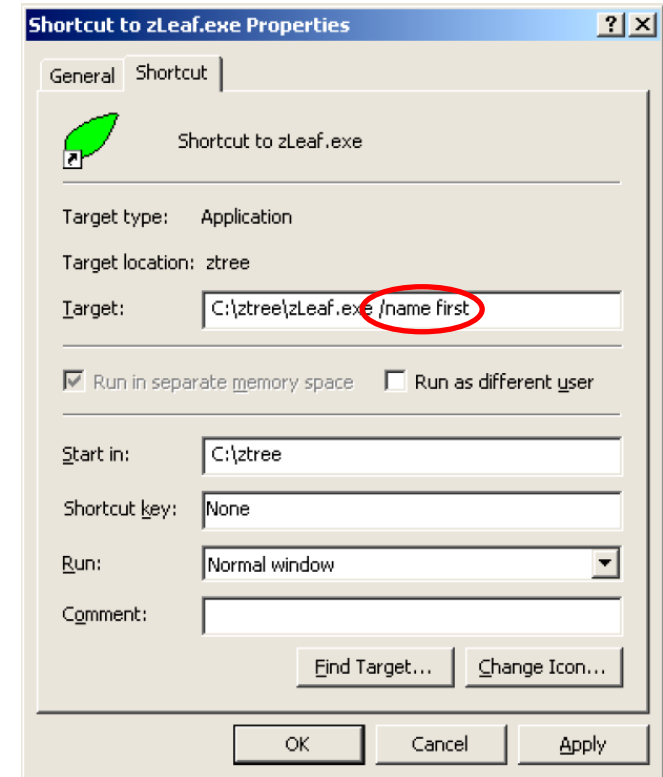
Profit display

- Add new box to Active screen of *Profit Display* stage
- Insert display items
 - Own contribution
 - Sum of all contributions
 - Subject's income for the period
- Insert "Continue" button
- Save the treatment (.ztt file)



Starting multiple z-leafs for testing

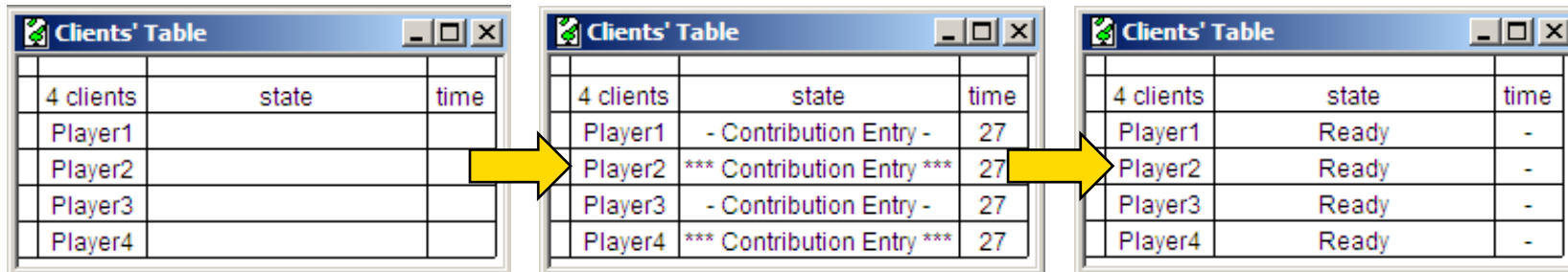
- Either create shortcuts:
- ...or use the following batch file:



```
@echo off
SETLOCAL ENABLEEXTENSIONS
SET /P leaves=How many zleafs do you want? %=%
for /L %%X IN (1,1,%leafs%) do (
start zleaf.exe /size 1024x768 /name Player%%X% /fontsize 11
PING 1.1.1.1 -n 1 -w 10 >NUL)
```

Advice for testing 1/2

- Switch between leafs using <ALT>-<TAB>
- Kill leafs using <ALT>-<F4>
- Display connected clients by clicking *Run-Clients Table*



The image shows three sequential screenshots of a window titled "Clients' Table". Each window contains a table with four columns: "4 clients", "state", and "time". The rows represent Player1, Player2, Player3, and Player4. Yellow arrows indicate the progression from the first to the second, and then to the third screenshot.

4 clients	state	time
Player1		
Player2		
Player3		
Player4		

4 clients	state	time
Player1	- Contribution Entry -	27
Player2	*** Contribution Entry ***	27
Player3	- Contribution Entry -	27
Player4	*** Contribution Entry ***	27

4 clients	state	time
Player1	Ready	-
Player2	Ready	-
Player3	Ready	-
Player4	Ready	-

- Pause time using <F12>
- Resume time using <Shift>-<F12>
- Stop testing by clicking *Run-Stop after this period*

Advice for testing 2/2

To test at your leisure:

- Make periods very short (e.g. 2 sec)
- Start program (<F5>)...
 - ...then immediately pause the time (<F12>)
- Test at will
- Resume/Pause to move to next period (<Shift>-<F12>)
- Click *Run-Stop after this period* to initiate end of testing
- End testing by clicking “OK” on all waiting screens

Resulting screen output

Period: 1 of 10 Time remaining [sec]: 27

Period: 1 of 10 Time remaining [sec]: 26

Your end
Your contribution to the

Your contribution to the project 7
Sum of all contributions 26
Your income in this period 23.4

subjects table

Period	Subject	Group	Profit	TotalProfit	Participate	EfficiencyFactor	Endowment	Contribution	TimeOK	ContributionEntryOK	SumC	N	TimeContinue	ProfitCalculationOK
1	1	1	23.4	23.4	1	1.6	20	7	99999	26	4	0		
1	2	1	27.4	27.4	1	1.6	20	3	99999	26	4	-		
1	3	1	19.4	19.4	1	1.6	20	11	99999	26	4	-		
1	4	1	25.4	25.4	1	1.6	20	5	99999	26	4	-		

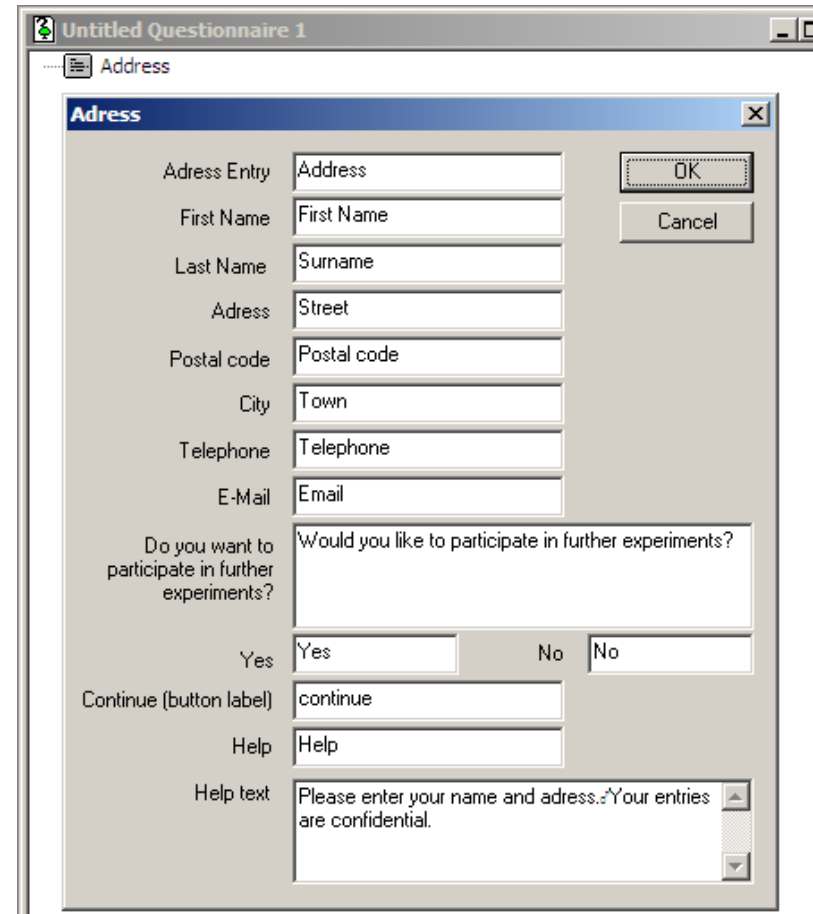
Resulting data output

- Output in YMMDD_hhmm.xls:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	090707_0824	1	globals	Period	NumPeriods	RepeatTreatment									
2	090707_0824	1	globals	1	10	0									
3	090707_0824	1	subjects	Period	Subject	Group	Profit	TotalProfit	Participate	EfficiencyFactor	Endowment	Contribution	TimeOK	ContributionEntryOK	SumC N Tim
4	090707_0824	1	subjects	1	1	1	19	19	1	1.6	20	13		29	30 4
5	090707_0824	1	subjects	1	2	1	30	30	1	1.6	20	2		26	30 4
6	090707_0824	1	subjects	1	3	1	24	24	1	1.6	20	8		24	30 4
7	090707_0824	1	subjects	1	4	1	25	25	1	1.6	20	7		20	30 4
8	090707_0824	1	globals	Period	NumPeriods	RepeatTreatment									
9	090707_0824	1	globals	2	10	0									
10	090707_0824	1	subjects	Period	Subject	Group	Profit	TotalProfit	Participate	EfficiencyFactor	Endowment	Contribution	TimeOK	ContributionEntryOK	SumC N Tim
11	090707_0824	1	subjects	2	1	1	28.4	47.4	1	1.6	20	4		29	31 4
12	090707_0824	1	subjects	2	2	1	25.4	55.4	1	1.6	20	7		27	31 4
13	090707_0824	1	subjects	2	3	1	12.4	36.4	1	1.6	20	20		25	31 4
14	090707_0824	1	subjects	2	4	1	32.4	57.4	1	1.6	20	0		23	31 4
15	090707_0824	1	globals	Period	NumPeriods	RepeatTreatment									
16	090707_0824	1	globals	3	10	0									
17	090707_0824	1	subjects	Period	Subject	Group	Profit	TotalProfit	Participate	EfficiencyFactor	Endowment	Contribution	TimeOK	ContributionEntryOK	SumC N Tim
18	090707_0824	1	subjects	3	1	1	19.6	67	1	1.6	20	14		28	34 4
19	090707_0824	1	subjects	3	2	1	26.6	82	1	1.6	20	7		22	34 4
20	090707_0824	1	subjects	3	3	1	25.6	62	1	1.6	20	8		20	34 4
21	090707_0824	1	subjects	3	4	1	28.6	86	1	1.6	20	5		17	34 4
22	090707_0824	1	summary	Period											
23	090707_0824	1	summary	1											
24	090707_0824	1	summary	2											
25	090707_0824	1	summary	3											
26	090707_0824	1	session	Subject	FinalProfit	ShowUpFee	ShowUpFeeInvested	MoneyAdded	MoneyToPay	MoneyEarned					
27	090707_0824	1	session	1	4.69	10	0	0	14.69	14.69					
28	090707_0824	1	session	2	5.74	10	0	0	15.74	15.74					
29	090707_0824	1	session	3	4.34	10	0	0	14.34	14.34					
30	090707_0824	1	session	4	6.02	10	0	0	16.02	16.02					

Creating new questionnaire

- Click *File-New Questionnaire*
- Click *Questionnaire-New Address Form*
 - Questions left empty will not be asked
 - May be entirely empty
 - Address form plus empty question form suffice to complete questionnaire
 - Run questionnaire (may be empty) at least once to write payment file

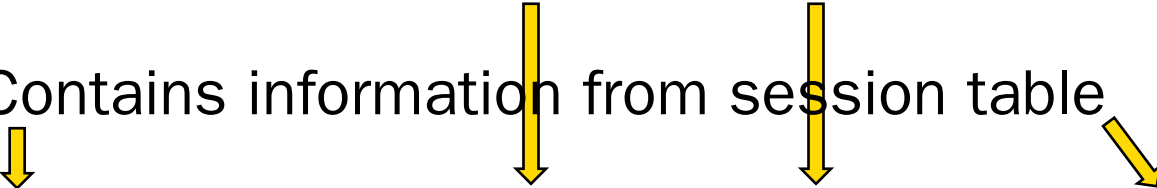


The screenshot displays a software window titled "Untitled Questionnaire 1" with a sub-window titled "Address". The "Address" form contains the following fields and options:

Address Entry	Address	OK
First Name	First Name	Cancel
Last Name	Surname	
Adress	Street	
Postal code	Postal code	
City	Town	
Telephone	Telephone	
E-Mail	Email	
Do you want to participate in further experiments?	Would you like to participate in further experiments?	
Yes	Yes	No No
Continue (button label)	continue	
Help	Help	
Help text	Please enter your name and address. Your entries are confidential.	

Payment file

- Created after treatment and questionnaire have been run
- Contains information from address form
- Contains information from session table



Subject	Computer	Interested	Name	Profit	Signature
1	Player1	OK	Jane Parker	21.52	
2	Player2	OK	Jim Smith	15.43	
3	Player3	OK	John Doe	20.00	
4	Player4	OK	Bill Farmer	17.23	
Experiment	C:\Institut\zTree\090728_1127.pay			74.18	

Part II

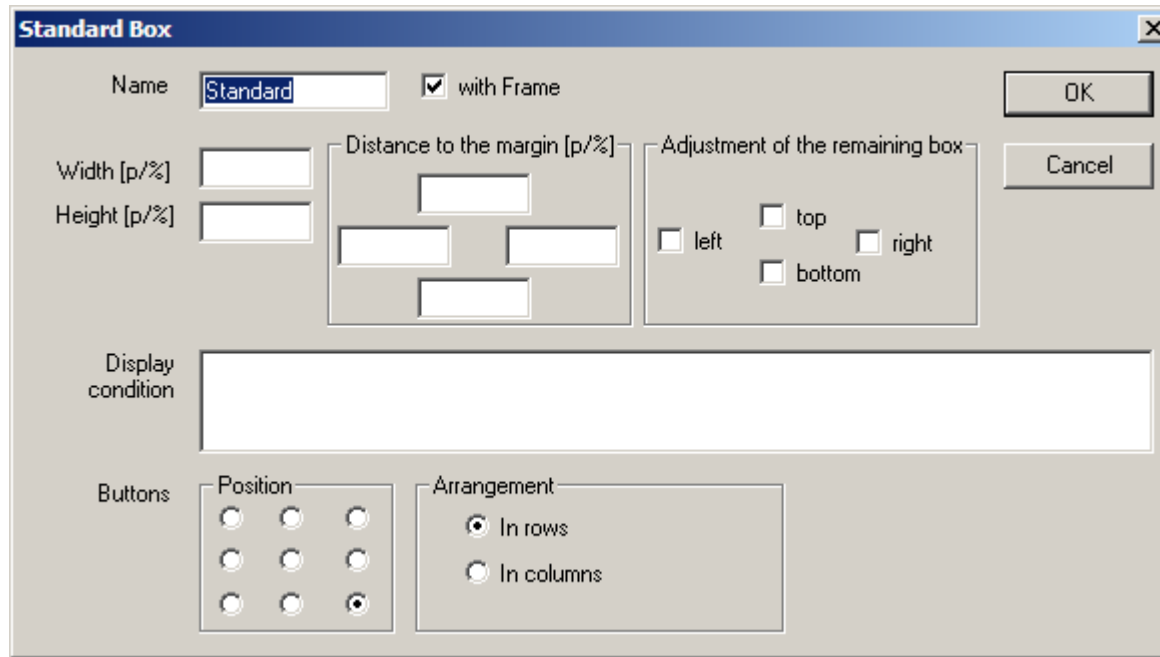
The Basics of z-Tree

Creating a stage

- Name is for documentation only; must be unique
- Start
- Timeout
- Active screen/Waiting screen/Use background
- Screen layout
 - Containers
 - Boxes
 - Absolute/relative size/position
 - Adjustment of the remaining box

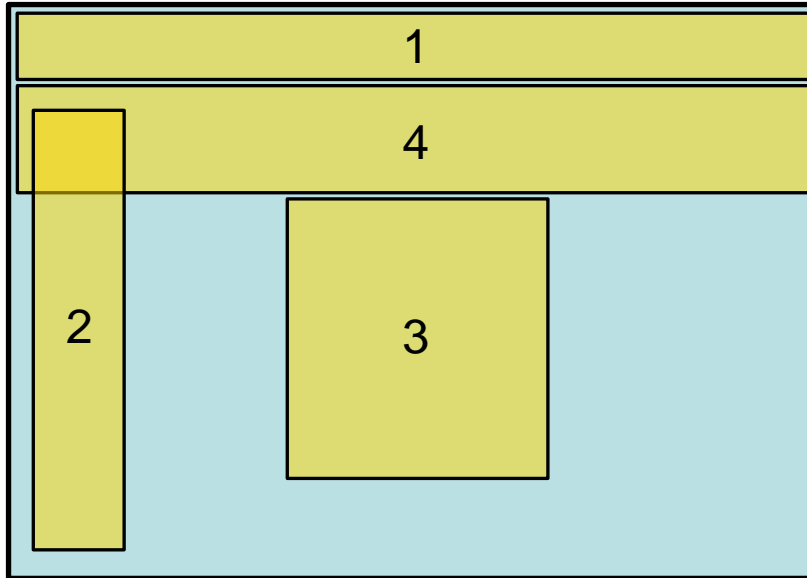
Screen layout

- Screen layout can be controlled in box dialog:



- See `Demo_ScreenLayout.ztt` for example layouts

Screen layout

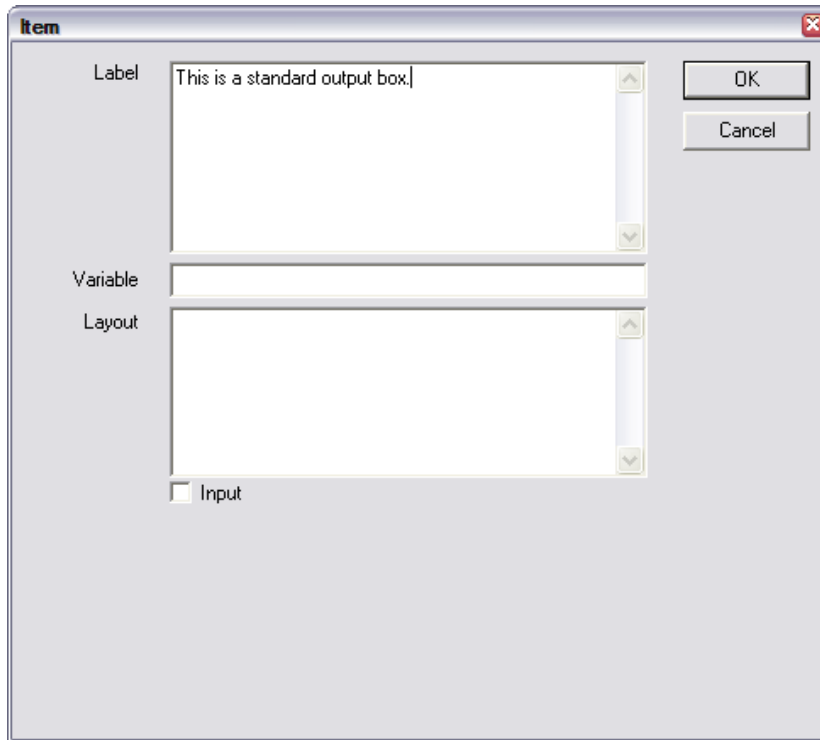


Name	1	<input checked="" type="checkbox"/> with Frame
Width [p/%]		Distance to the margin [p/%]
Height [p/%]	10%	0p
		Adjustment of the remaining box
		<input type="checkbox"/> left <input checked="" type="checkbox"/> top <input type="checkbox"/> right
		<input type="checkbox"/> bottom
Name	2	<input checked="" type="checkbox"/> with Frame
Width [p/%]	50p	Distance to the margin [p/%]
Height [p/%]		20p
		20p
		20p
		Adjustment of the remaining box
		<input type="checkbox"/> left <input type="checkbox"/> top <input type="checkbox"/> right
		<input type="checkbox"/> bottom
Name	3	<input checked="" type="checkbox"/> with Frame
Width [p/%]	30%	Distance to the margin [p/%]
Height [p/%]	50%	
		Adjustment of the remaining box
		<input type="checkbox"/> left <input type="checkbox"/> top <input type="checkbox"/> right
		<input checked="" type="checkbox"/> bottom
Name	4	<input checked="" type="checkbox"/> with Frame
Width [p/%]		Distance to the margin [p/%]
Height [p/%]		
		Adjustment of the remaining box
		<input type="checkbox"/> left <input type="checkbox"/> top <input type="checkbox"/> right
		<input type="checkbox"/> bottom

Examples of simple functions

- `result = sum (same (Group), x);`
- `result = count (i == 5, y);`
- `result = average (i != 5 & i > 2, y);`
- `result = if (k < 5 | k >= 10, 1, 10);`
- `result = abs (c + pi());`
- `result = round (a, 0.2);`
- `result = roundup (random() * 5, 1);`

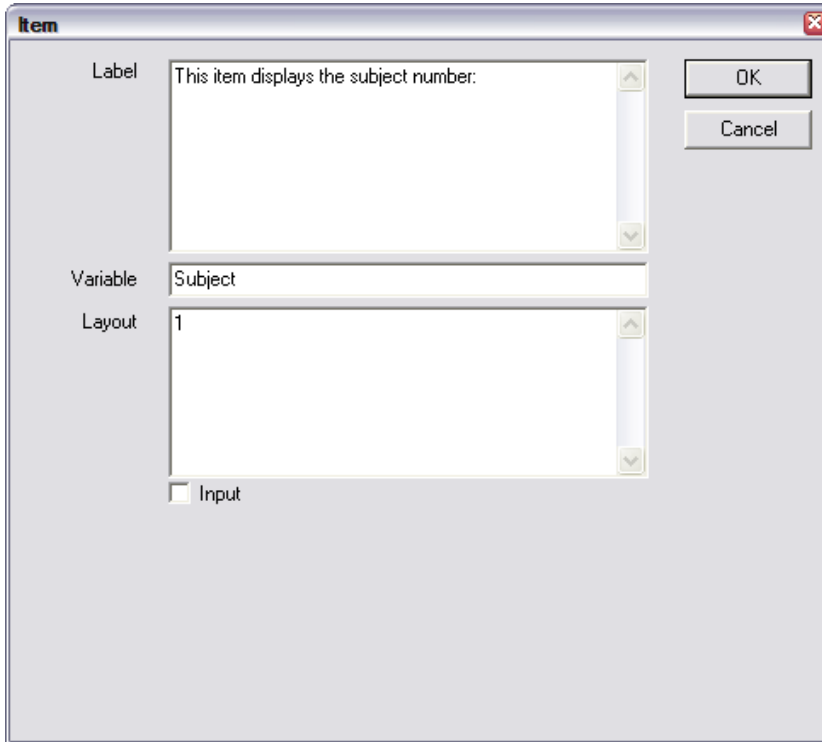
Items: Text output



- Result:

This is a standard output box.

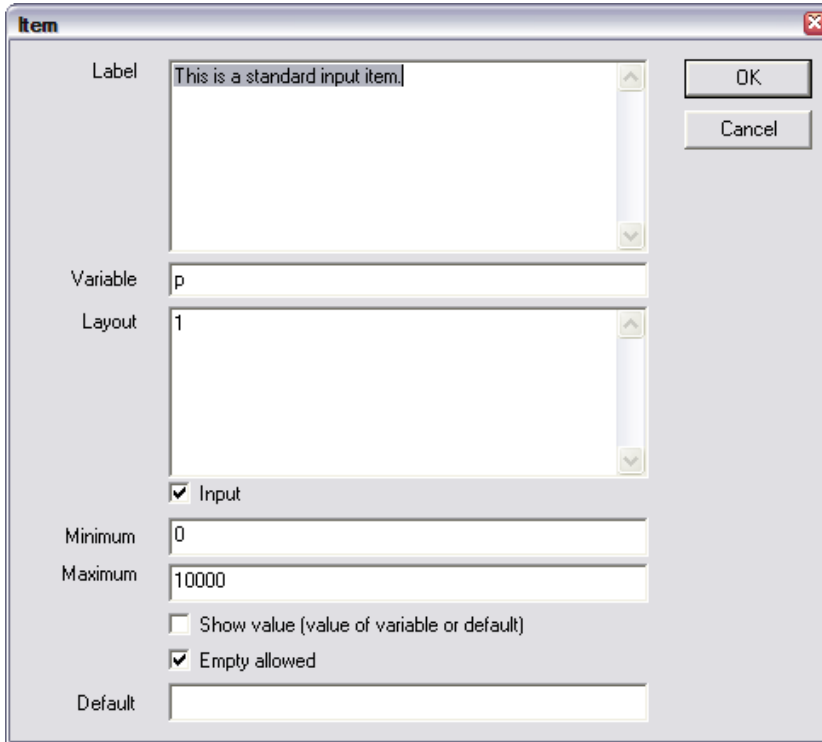
Items: Variable output



- Result:

This item displays the subject number: 1

Items: Variable input



Item

Label: This is a standard input item.

Variable: p

Layout: 1

Input

Minimum: 0

Maximum: 10000

Show value (value of variable or default)

Empty allowed


Default:

OK

Cancel

■ Result:

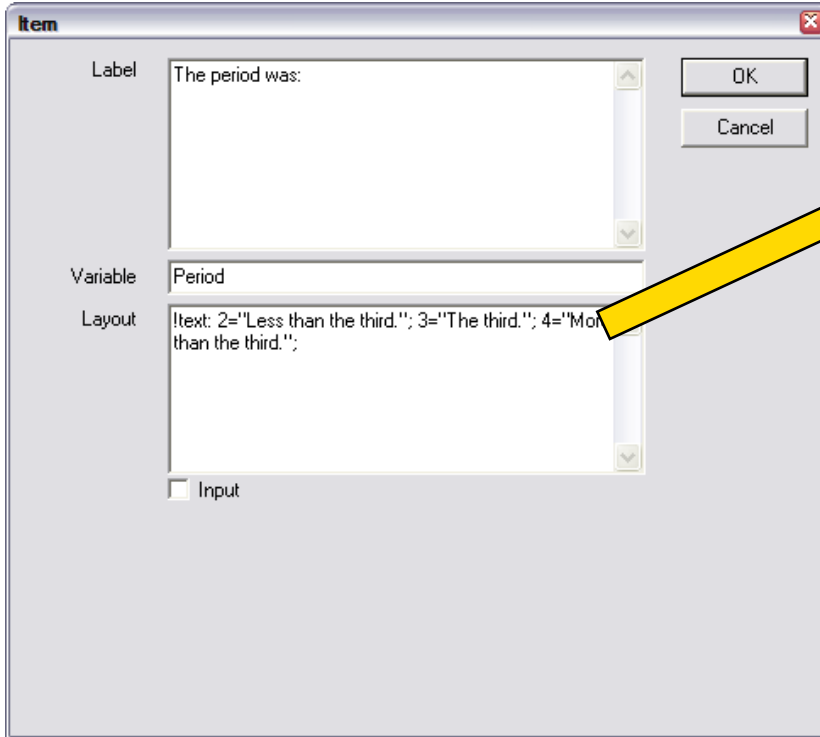
This is a standard input item.



Part III

Advanced Topics in z-Tree

Conditional formatting



Item

Label: The period was:

Variable: Period

Layout: !text: 2="Less than the third."; 3="The third."; 4="More than the third."

Input

OK

Cancel

Layout:

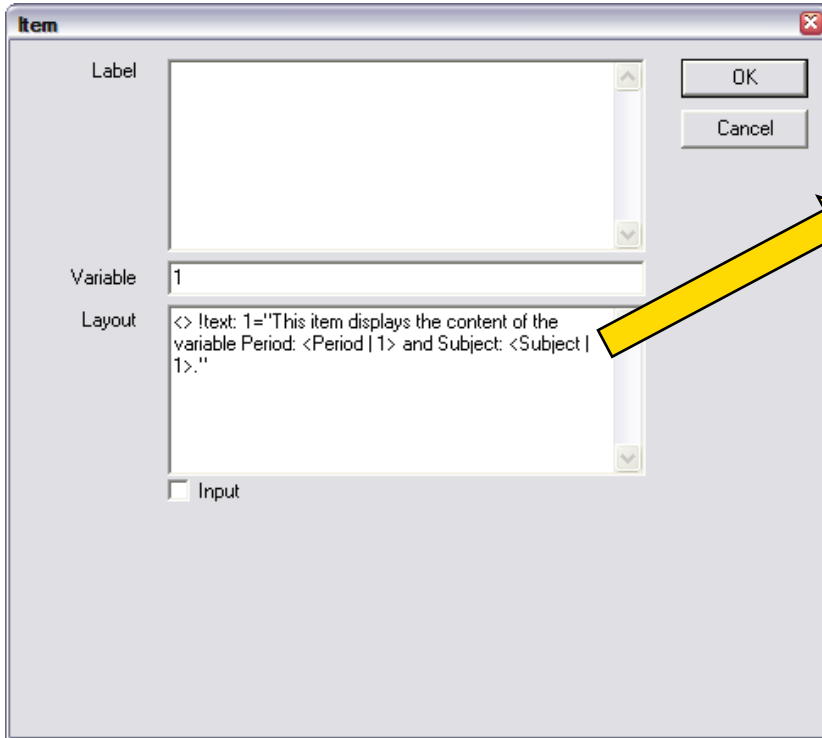
!text: 2="Less than the third."; 3="The third."; 4="More than the third.";

■ Results:

Period 1+2: The period was: Less than the third. Period 3: The period was: The third.

Period 4+: The period was: More than the third.

Inserting variables into text



The screenshot shows a dialog box titled "Item" with the following fields:

- Label:** An empty text area.
- Variable:** A dropdown menu showing "1".
- Layout:** A text area containing the code: `<> !text: 1="This item displays the content of the variable Period: <Period | 1> and Subject: <Subject | 1>."` A yellow arrow points from this text to the "Layout" field.
- Input:** An unchecked checkbox.
- Buttons:** "OK" and "Cancel" buttons.

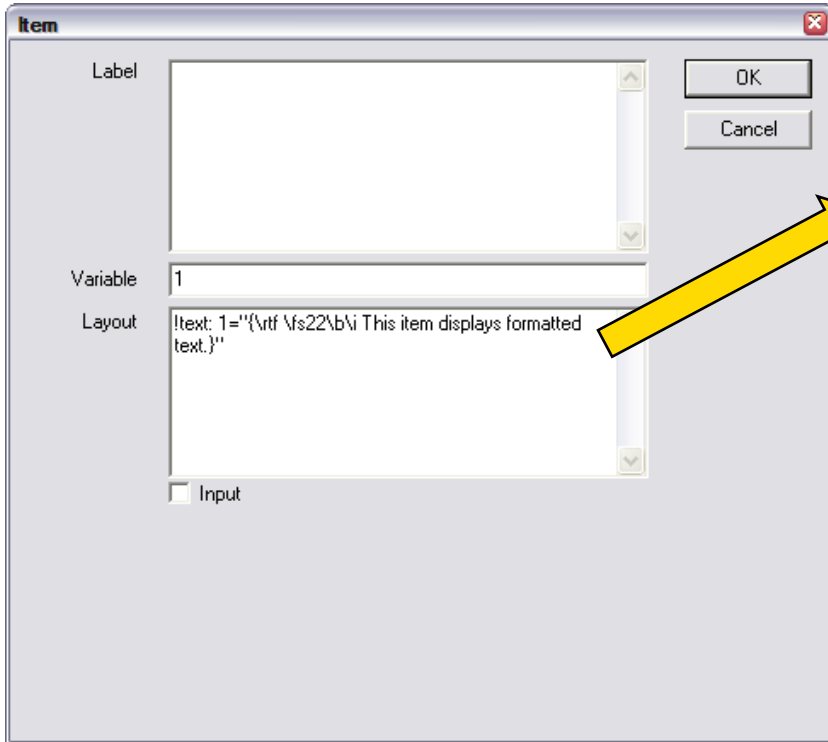
Layout:

`<> !text: 1="This item displays the content of the variable Period: <Period | 1> and Subject: <Subject | 1>."`

■ Result:

This item displays the content of the variable Period: 1 and Subject: 2.

Formatted text



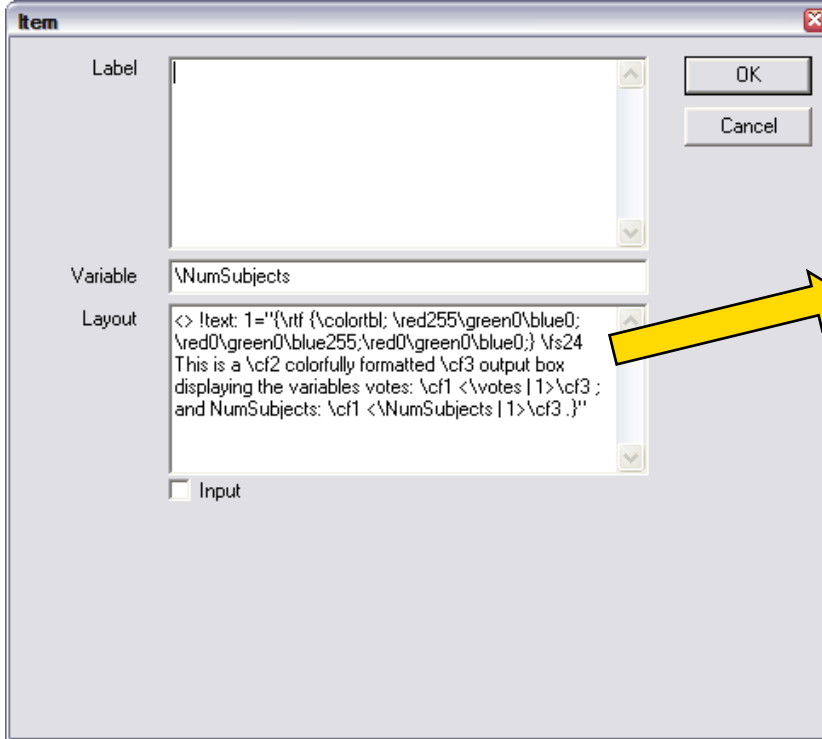
Layout:

!text: 1="{\rtf \fs22\b\i This item displays formatted text.}"

- **Result:**

This item displays formatted text.

Variables embedded in colored text



The 'Item' dialog box shows the following fields:

- Label:** (Empty)
- Variable:** \NumSubjects
- Layout:**

```
<> !text: 1="{\rtf {\colortbl; \red255\green0\blue0; \red0\green0\blue255;\red0\green0\blue0;} \fs24 This is a \cf2 colorfully formatted \cf3 output box displaying the variables votes: \cf1 <\votes | 1>\cf3 ; and NumSubjects: \cf1 <\NumSubjects | 1>\cf3 .}"
```
- Input:**

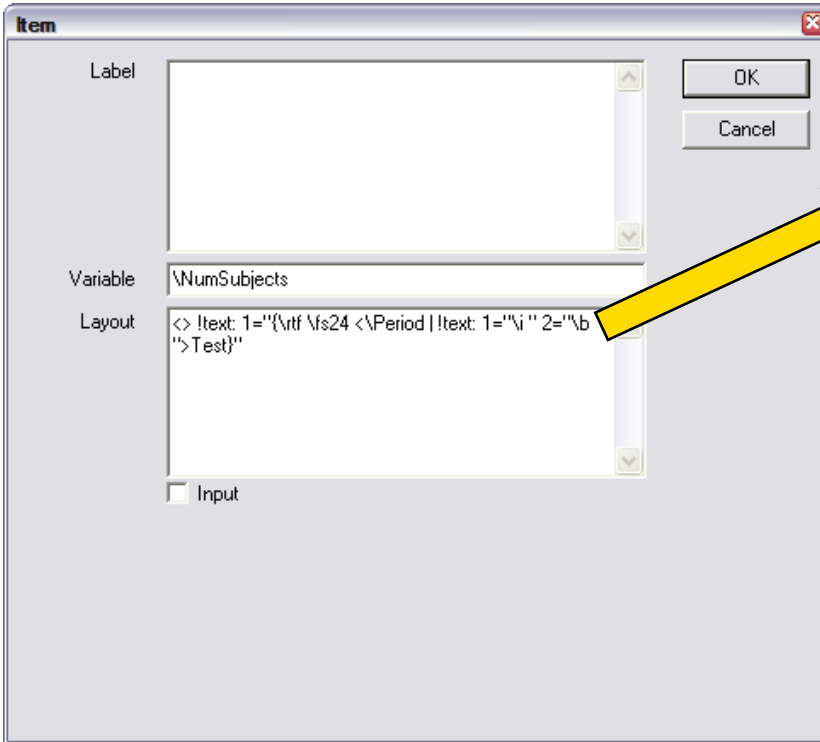
Layout:

```
<> !text: 1="{\rtf {\colortbl; \red255\green0\blue0; \red0\green0\blue255;\red0\green0\blue0;} \fs24 This is a \cf2 colorfully formatted \cf3 output box displaying the variables votes: \cf1 <\votes | 1>\cf3 ; and NumSubjects: \cf1 <\NumSubjects | 1>\cf3 .}"
```

■ Result:

This is a **colorfully formatted** output box displaying the variables votes: **0**; and NumSubjects: **1**.

Conditional formatting



Item

Label

Variable: \NumSubjects

Layout: <> !text: 1='{\rtf \fs24 <\Period | !text: 1='\i ' 2='\b '>Test}'

Input

OK

Cancel

Layout:

```
<> !text: 1="{\rtf \fs24 <\Period | !text:
1="\i " 2="\b ">Test}"
```

- Result:

Period 1: *Test*

Period 2+: **Test**

Alternative formats

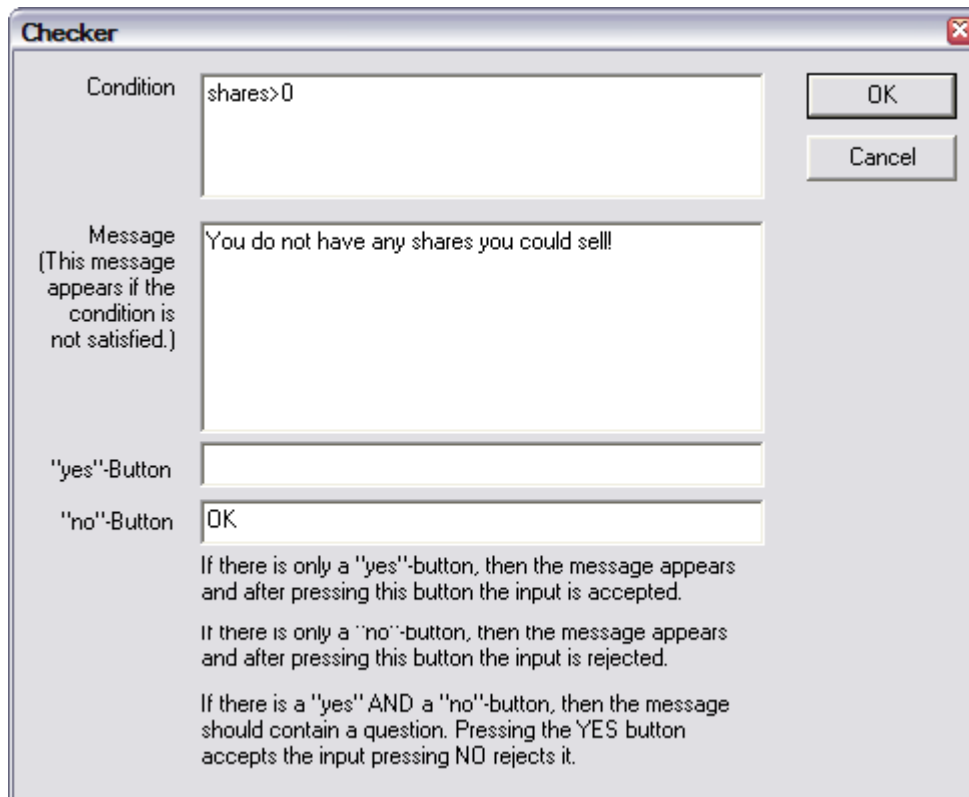
Examples of item layouts

Layout	input variable	output variable
2	<input type="text" value="6"/>	<input type="text" value="6"/>
!radio: 1 = "86.8"; 24 = "102.8";	<input checked="" type="radio"/> 86.8 <input type="radio"/> 102.8	<input checked="" type="radio"/> 86.8 <input type="radio"/> 102.8
!radioline: 0="zero";5="five"; 6;	zero <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> five	zero <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input type="radio"/> five
!slider: 0 = "A"; 100 = "B"; 101;	A <input type="range" value="50"/> B	A <input type="range" value="50"/> B
!scrollbar: 0="L";100= "R";101;	L <input type="range" value="50"/> R	L <input type="range" value="50"/> R
!checkbox:1="check me";	<input checked="" type="checkbox"/> check me	<input checked="" type="checkbox"/> check me
!text: 1 = "one"; 2 = "two"; 3 = "three"; 4 = "four"; 5 = "five"; 6 = "six"; 7 = "seven"; 8 = "eight"; 9 = "nine"; 10 = "ten";	<input type="text" value="seven"/>	<input type="text" value="seven"/>
!button: 1 = "accept"; 0 = "reject";	<input type="button" value="accept"/> <input type="button" value="reject"/>	<input type="text" value="accept"/>

Source: Reference manual, p. 30.

Checking entries for correctness

- Select a button
- Click *Treatment-New Checker...*



The screenshot shows a dialog box titled "Checker" with a close button (X) in the top right corner. The dialog is divided into several sections:

- Condition:** A text input field containing "shares>0".
- Message:** A text area containing "You do not have any shares you could sell!". To the left of this area is the text "(This message appears if the condition is not satisfied.)".
- "yes"-Button:** An empty text input field.
- "no"-Button:** A text input field containing "OK".

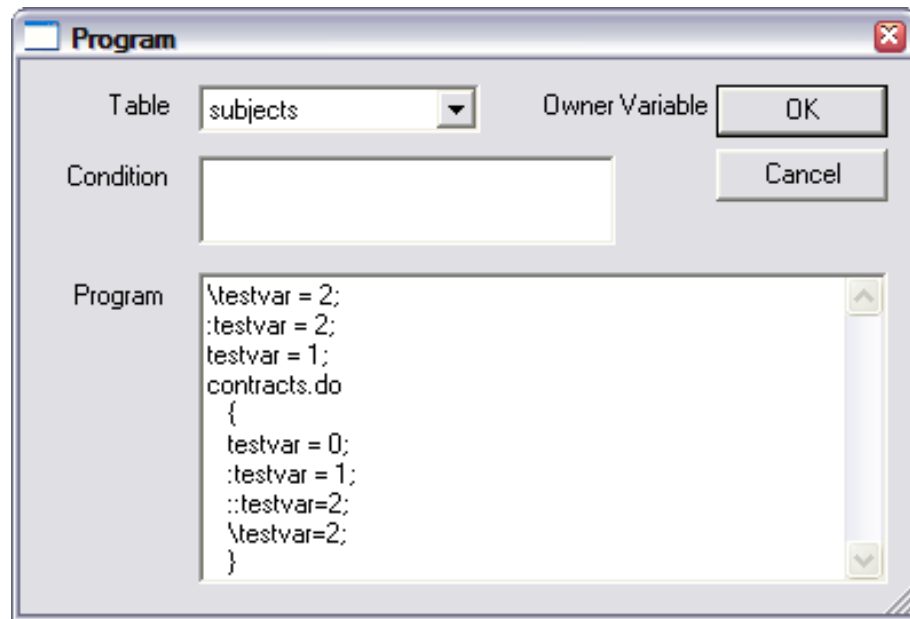
Below the input fields, there are three paragraphs of explanatory text:

- "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."

On the right side of the dialog, there are two buttons: "OK" and "Cancel".

Programs, tables and scope operators

- Programs run in a table
- Programs can contain commands running in different table
- Structure here is:
 - globals
 - subjects
 - contracts



Scope operators:

- : one step up
- \ all steps up to „globals“ table

Programs, tables and scope operators

Programs run for all records in a table (unless restricted by conditions)

Command run in table „subjects“

The screenshot shows a dialog box titled "Program" with the following fields and content:

- Table:** A dropdown menu set to "globals".
- Owner Variable:** An empty text field.
- Condition:** An empty text field.
- Program:** A text area containing the following code:


```
//Randomly assigns endowment type
endowmenttype=1;
//Repeats as often as there are endowment types

while (endowmenttype<=endowmenttypes)
{
  i=1;
  //Repeats as often as one endowment type is assigned to a trader (e.g. 10 traders, 2 types = 5 times)
  while (k<=endowmentspertype)
  {
    j=roundup(random()*((NumSubjects-(endowmenttype-1)*endowmentspertype-(i-1)),1);
    k=1;
    l=0;
    done=0;
    while (k<=NumSubjects&done==0)
    {
      if (subjects.find(Subject==k,InitialMoney)==0) {l=l+1;}
      if (l==1)
      {
        subjects.do
        {
          if (Subject==k)
          {
            InitialMoney=endowments.find(type==endowmenttype,cash);
            InitialStock=endowments.find(type==endowmenttype,stock);
            \done=1;
          }
        }
        k=k+1;
      }
      i=i+1;
    }
    endowmenttype=endowmenttype+1;
  }
}
```

Annotations in the image:

- Two yellow arrows point from the text "Programs run for all records in a table (unless restricted by conditions)" to the "Table" and "Condition" fields.
- Two yellow arrows point from the text "Command run in table „subjects“" to the code lines `subjects.find(Subject==k,InitialMoney)==0` and `subjects.do`.
- A yellow box labeled "Variables in table „globals“" has red arrows pointing to `endowmenttype=1;`, `endowmenttypes`, `endowmentspertype`, and `NumSubjects`.
- A yellow box labeled "Variable in table „subjects“" has black arrows pointing to `subjects.find(Subject==k,InitialMoney)==0` and `subjects.do`.

Programs, tables and scope operators

- Example: (Program runs in table A)

$$x = v + B.sum (v * :v - C.product (v - :v - ::v))$$

A A B A C B A

- Function same() as a special case
- If variable only defined in one table, scope operator may be omitted (dangerous!)
- Last period's tables: OLDsubjects, etc.

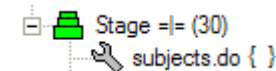
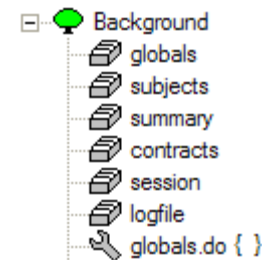
The complete scope reference

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	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	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

Source: Reference manual, p. 59.

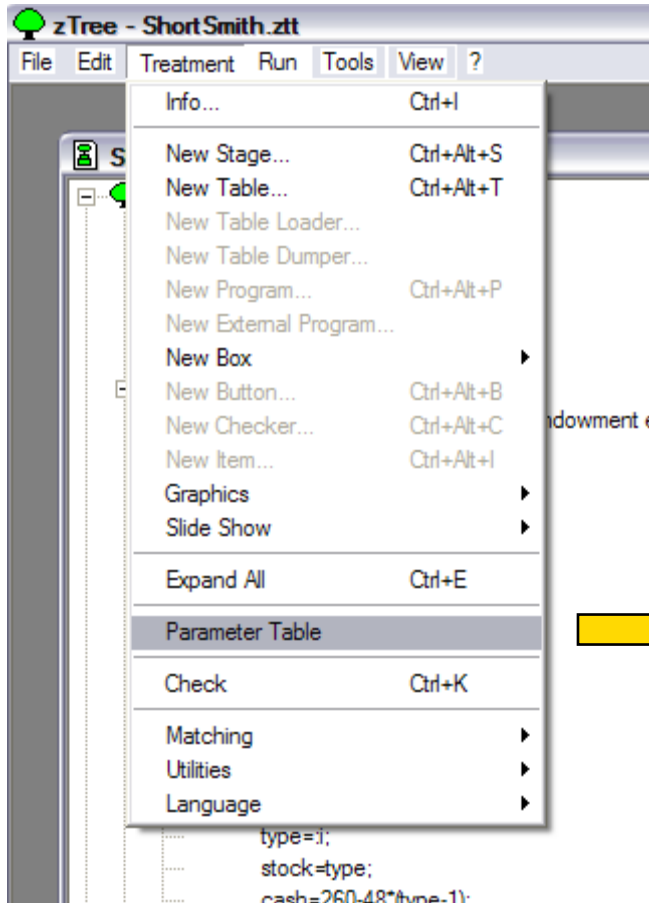
Running programs

- Program in the background
 - Runs after the setting of standard variables
- Program in a stage
 - Runs at beginning of stage, before checking Participate
- Program in a button
 - Program in subjects table runs only for the subject pressing the button
 - Runs after checkers
- Program started by later () do/repeat { } command
 - (Re-)Runs after pre-specified time



Program later (5) do {a=a+1;}

Parameter table



1. Specific parameters
(after programs in the background)

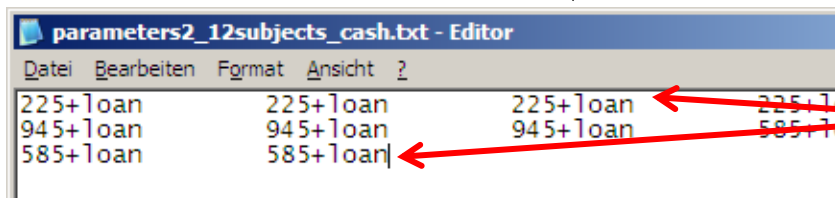
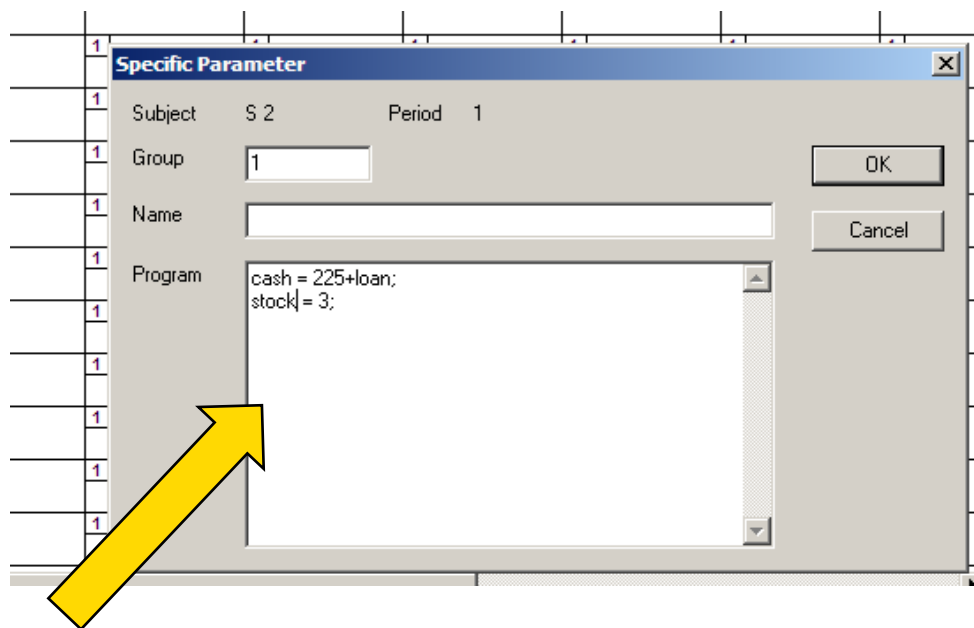
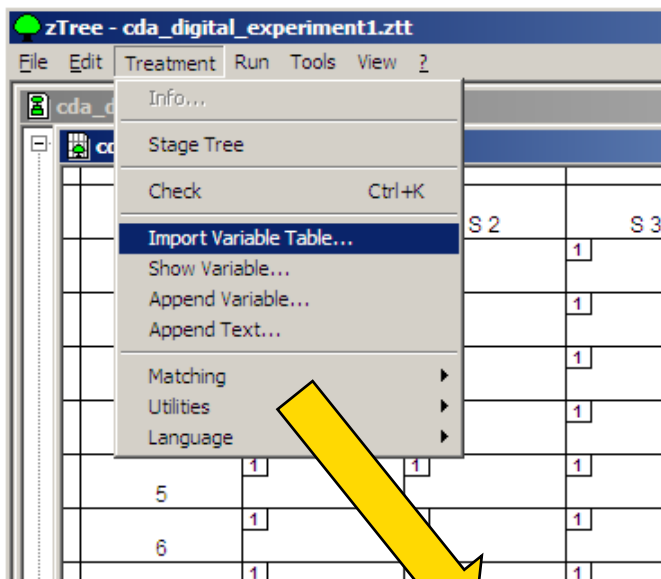
2. Role parameters

	S1	S2	S3	S4
1	1	1	1	1
2	1	1	1	1

3. Period parameters & prompt

Parameter table variable import

- Variables can be imported from ASCII file (one-by-one)
- Allows parameter customization before session



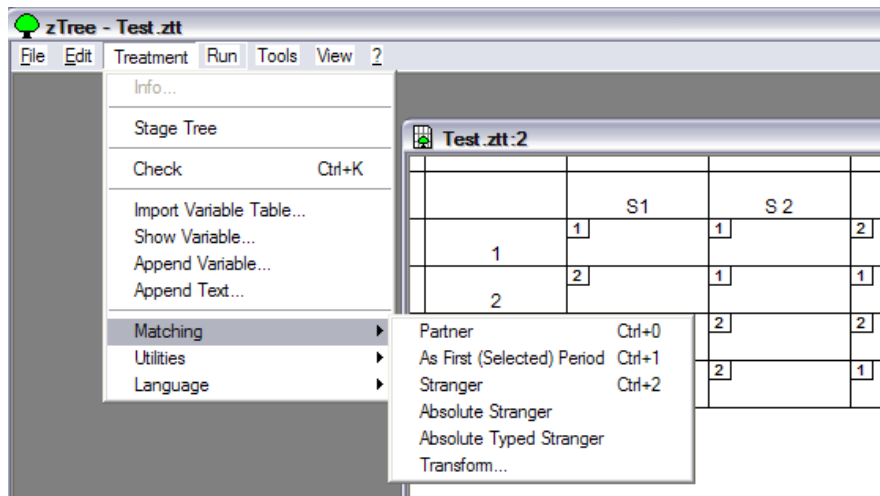
Tab-separated values

Period structure

1. Setting of standard variables
2. Programs in the Background
3. Specific parameter programs
4. Role parameter programs
5. Period parameter programs
6. Programs at the beginning of a stage
7. Programs in buttons (when clicked)
 - Delayed sub-programs: *later () do/repeat { }*

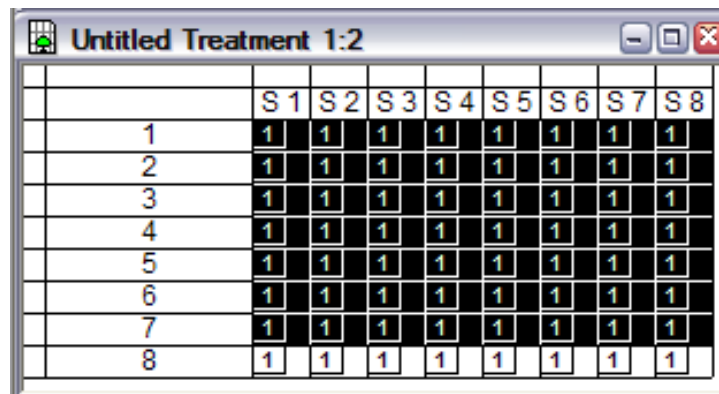
Groups

- Groups are defined in Specific parameters in Parameter table
- Groups can be modified using variable *Group* in subjects table
- Automatic group matching:



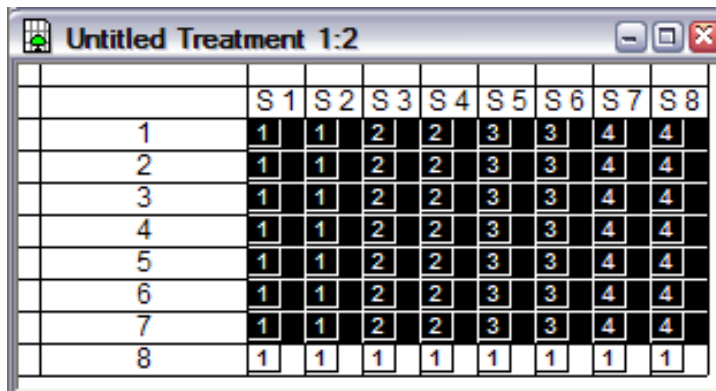
Group matching

- Select desired periods in parameter table:

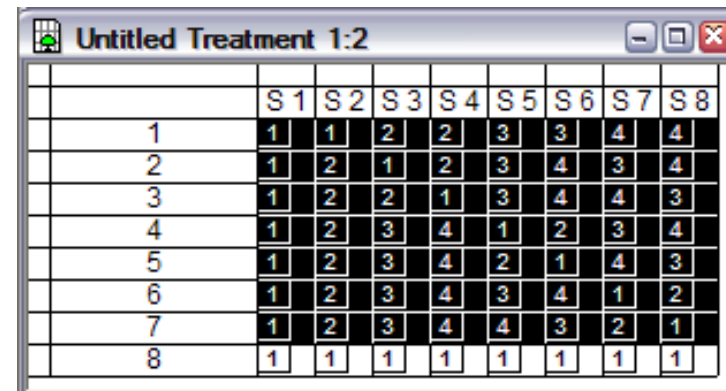


	S 1	S 2	S 3	S 4	S 5	S 6	S 7	S 8
1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1

- Choose matching procedure from *Treatment* menu:
 - Partner:
 - Absolute stranger:



	S 1	S 2	S 3	S 4	S 5	S 6	S 7	S 8
1	1	1	2	2	3	3	4	4
2	1	1	2	2	3	3	4	4
3	1	1	2	2	3	3	4	4
4	1	1	2	2	3	3	4	4
5	1	1	2	2	3	3	4	4
6	1	1	2	2	3	3	4	4
7	1	1	2	2	3	3	4	4
8	1	1	1	1	1	1	1	1



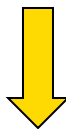
	S 1	S 2	S 3	S 4	S 5	S 6	S 7	S 8
1	1	1	2	2	3	3	4	4
2	1	2	1	2	3	4	3	4
3	1	2	2	1	3	4	4	3
4	1	2	3	4	1	2	3	4
5	1	2	3	4	2	1	4	3
6	1	2	3	4	3	4	1	2
7	1	2	3	4	4	3	2	1
8	1	1	1	1	1	1	1	1

z-Tree programming with Excel

- Use MS Excel to generate repetitive code:

	A	B	C	D	E	F
1	Period	Dividend	Code			
2	1	20	if (Period==1) {dividend=20;}			
3	2	30	if (Period==2) {dividend=30;}			
4	3	10	if (Period==3) {dividend=10;}			

The formula bar for cell D2 shows: `= " if (Period==" &A2&" " &B2&" " &";)"`



Program

Table: Owner Variable:

Condition:

Program:

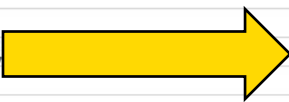
```
if (\timeseries==1)
{
  if (Period==1) {price=9;}
  if (Period==2) {price=39;}
  if (Period==3) {price=83;}
  if (Period==4) {price=23;}
  if (Period==5) {price=19;}
  if (Period==6) {price=25;}
  if (Period==7) {price=14;}
  if (Period==8) {price=37;}
}
```

Buttons: OK, Cancel

z-Tree programming with Excel

- Complex code generation:

A	B	C
1	Time	Price
2	40	365,29
3		later (round(40/daylengthbase*daylength,1)) do
4		{
5		signals.new
6		{
7		signal=365;
8		signaltime=gettime()-summary.find(same(Period),timeperiodstarted);
9		hour=\daybeginhour+rounddown(\dayhours*(signaltime/\daylength),1);
10		minute=rounddown((signaltime-(hour-\daybeginhour)/\dayhours*\daylength)/\daylength,1);
11		weekday=1;
12		}
13	72	368,33
14		later (round(72/daylengthbase*daylength,1)) do
15		{
16		signals.new
17		{
18		signal=368;
19		signaltime=gettime()-summary.find(same(Period),timeperiodstarted);
20		hour=\daybeginhour+rounddown(\dayhours*(signaltime/\daylength),1);
21		minute=rounddown((signaltime-(hour-\daybeginhour)/\dayhours*\daylength)/\daylength,1);
22		weekday=1;
23		}
24	85	364,90
25		later (round(85/daylengthbase*daylength,1)) do
26		{
27		signals.new
28		{
29		signal=365;
30		signaltime=gettime()-summary.find(same(Period),timeperiodstarted);
31		hour=\daybeginhour+rounddown(\dayhours*(signaltime/\daylength),1);
32		minute=rounddown((signaltime-(hour-\daybeginhour)/\dayhours*\daylength)/\daylength,1);



```

zTree - [#daynight_baseline_overnight.ztt]
File Edit Treatment Run Tools View ?

StartingScreen =(-1)N
TradeScreenWE =!(welength)A
TradeScreenNight =!(nightlength)A
TradeScreenDay =!(daylength)A
summary.do {timeperiodstartedprevious=timeperiodstarted; ...}
globals(Period==1).do { //Generates Signals Period 01 ... }
globals(Period==2).do { //Generates Signals Period 02 ... }
globals(Period==3).do { //Generates Signals Period 03 ... }
globals(Period==4).do { //Generates Signals Period 04 ... }
globals(Period==5).do { ... }
//Generates Signals Period 05
summary.do{dividende=1;}
//Writes new signals into table signals at the time specified in the later function
later (round(31/daylengthbase*daylength,1)) do
{
signals.new
{
signal=384;
signaltime=gettime()-summary.find(same(Period),timeperiodstarted);
hour=\daybeginhour+rounddown(\dayhours*(signaltime/\daylength),1);
minute=rounddown((signaltime-(hour-\daybeginhour)/\dayhours*\daylength)/\daylength,1);
weekday=5;
}
}
later (round(48/daylengthbase*daylength,1)) do
{
signals.new
{
signal=393;
signaltime=gettime()-summary.find(same(Period),timeperiodstarted);
hour=\daybeginhour+rounddown(\dayhours*(signaltime/\daylength),1);
minute=rounddown((signaltime-(hour-\daybeginhour)/\dayhours*\daylength)/\daylength,1);
weekday=5;
}
}
later (round(55/daylengthbase*daylength,1)) do
{
signals.new
{
signal=378;
signaltime=gettime()-summary.find(same(Period),timeperiodstarted);
hour=\daybeginhour+rounddown(\dayhours*(signaltime/\daylength),1);
minute=rounddown((signaltime-(hour-\daybeginhour)/\dayhours*\daylength)/\daylength,1);
weekday=5;
}
}

```

Data import and export 1/2

- Tables can be imported and exported from/to ASCII file
- New (Version 3.3.0) stage tree elements

- Table dumper:
Allows exporting
table to ASCII
file

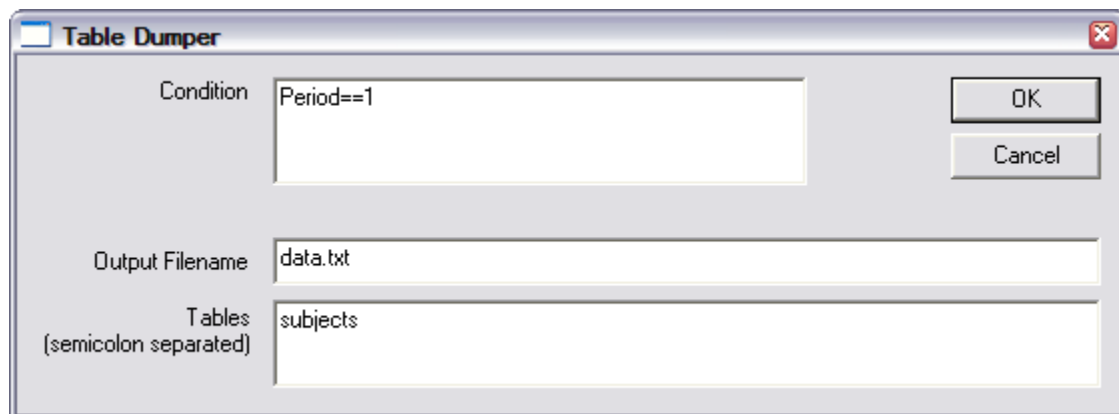


Table Dumper

Condition: Period==1

Output Filename: data.txt

Tables (semicolon separated): subjects

OK Cancel

- Table loader:
Allows importing
table from ASCII
file (append or
replace)

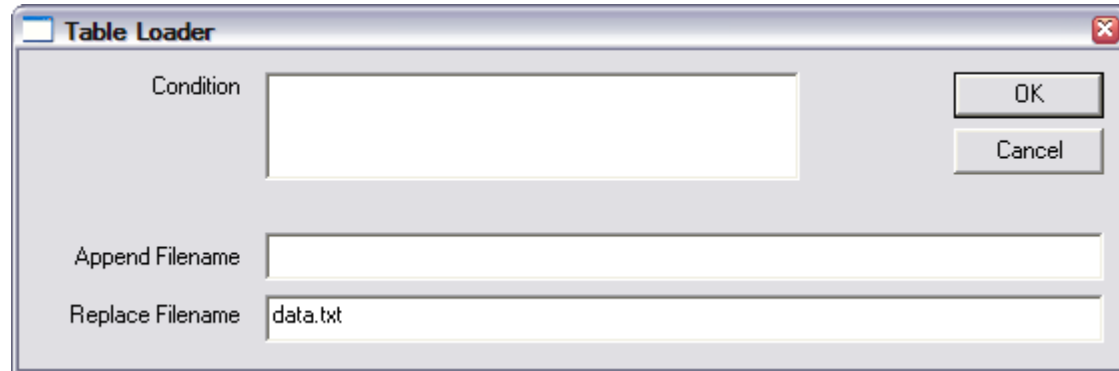


Table Loader

Condition:

Append Filename:

Replace Filename: data.txt

OK Cancel

Data import and export 2/2

Demo_DataFileRW.ztt

- Table dumper:

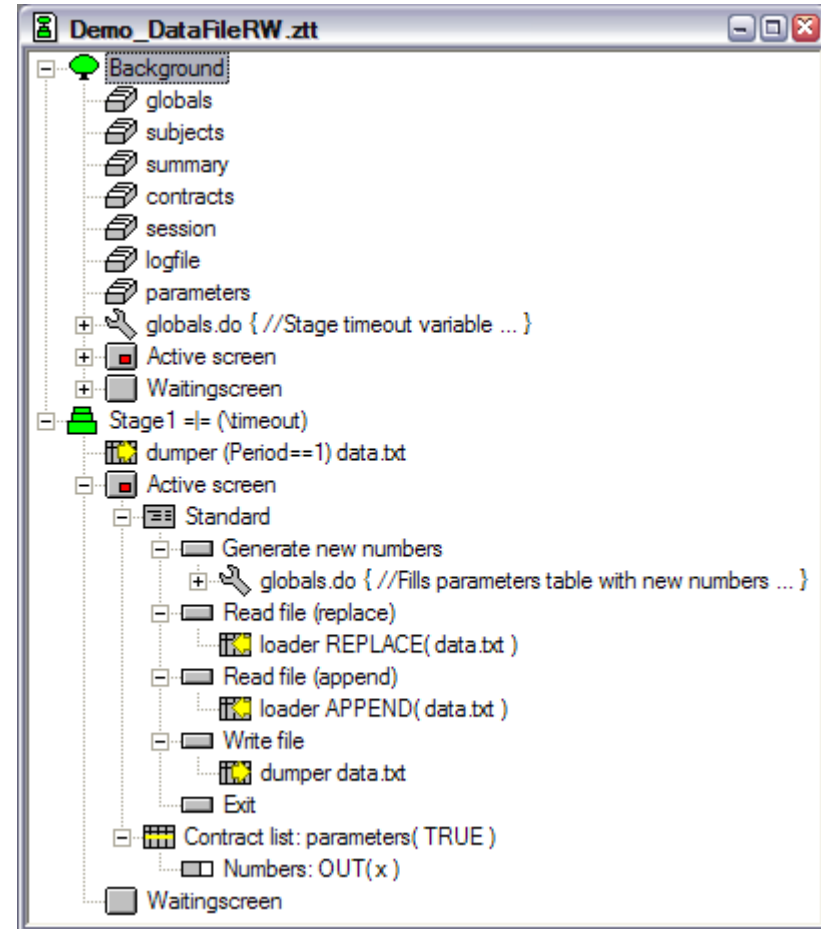
parameters	Period	x
parameters	1	68
parameters	1	3
parameters	1	94

- Table loader (replace):

Replaces table records from first downwards with data in ASCII file

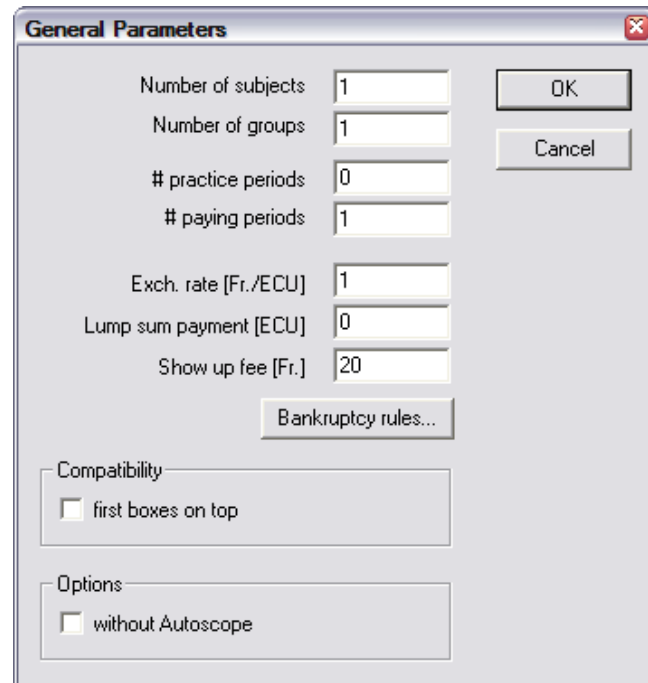
- Table loader (append):

Adds new table records after last one



Calculation of payouts: Background

- # practice periods
 - Periods without effect on profit
- # paying periods
 - Periods with profit
- Exchange rate
 - Value in real currency of 1 unit of the experimental currency (ECU)
- Lump sum payment
 - Value in ECU added to TotalProfit at the beginning of period 1 of the treatment
- Show up fee
 - Value in real currency added to subject payout at the beginning of period 1 of the session



General Parameters

Number of subjects	1
Number of groups	1
# practice periods	0
# paying periods	1
Exch. rate [Fr./ECU]	1
Lump sum payment [ECU]	0
Show up fee [Fr.]	20

Buttons: OK, Cancel, Bankruptcy rules...

Compatibility: first boxes on top

Options: without Autoscope

Calculation of payouts: Bankruptcy

Bankruptcy Rules ✕

Text "Do you invest your show up fee ?"

You have made a loss. Do you want to invest your show-up fee to compensate this loss?

"yes" Show up fee is invested and experiment goes on

"no" Message "BankruptShowupNo" appers in client's table*

Text "Do you want to go on"

You have made a loss. Do you want to continue?

"yes" Message "BankruptMoreYes" appers in client's table*

"no" Message "BankruptMoreNo" appers in client's table*

Text "Please wait until the experimenter unlocks your PC."

Please wait until the experimenter allows you to continue.

* The experimenter can either
 - give permission to go on (give a credit)
 - replace the subject (nullify payoffs)

Calculation of payouts: Profit/TotalProfit

Table: subjects

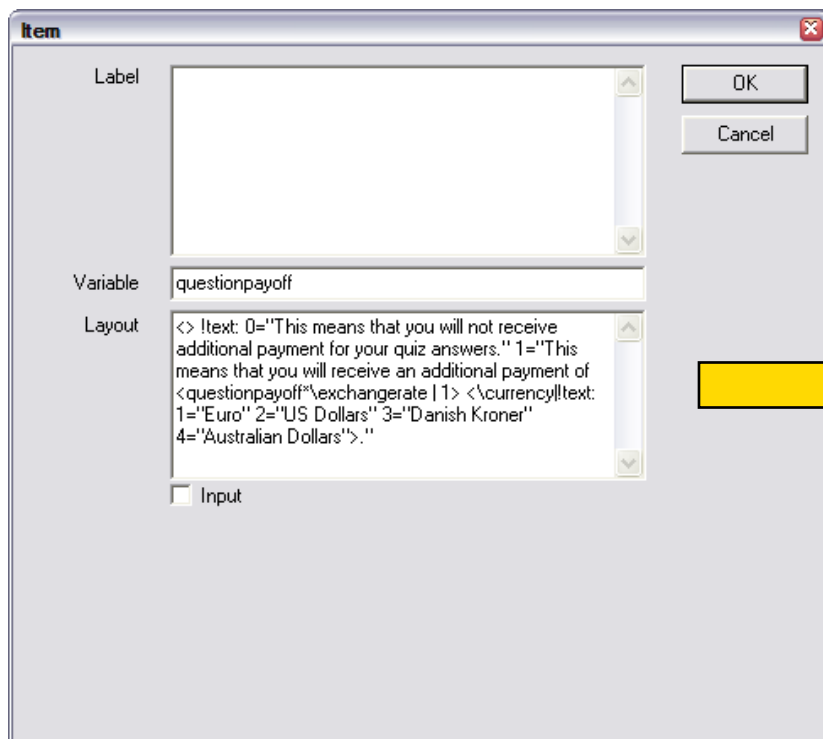
- Profit
 - Contains profit in a period, needs to be calculated
- TotalProfit
 - Contains total profit up to period t , calculated automatically as the sum of the variables *Profit* of periods 1 to $(t - 1)$.

Table: session

- FinalProfit
 - Subject's profit excluding the show-up fee
- MoneyAdded
 - Money added to a subject who faced bankruptcy but was allowed to continue
- MoneyToPay
 - Equals *FinalProfit* plus *ShowUpFee* plus *MoneyAdded*

Calculation of payouts: Good advice

- Set exchange rate in Background to 1
- Create variables in table globals.
 - exchangerate: The exchange rate in Real CU/ECU
 - currency: A dummy variable for currency, e.g. 1 for EUR, 2 for USD...



Only one place for changing these parameters (in the program in the globals table)!

Questionnaires: Overview

- Run after ≥ 1 treatment (sets number of subjects)
- Address form needed to write payment file (delete components by deleting caption in address form)
- Question forms used for:
 - Solicitation of feedback
 - Display of results
- Questionnaire may be empty

Questionnaires: Using the session table

Conditioning on variables in session table (*Participate*)

- Run treatment writing variables into session table (*FinalProfit*, *MoneyAdded*, *ShowUpFee*, *MoneyToPay* and *MoneyEarned* are always available)
- Access variables in session table in question form definition in questionnaire
- Use *Participate* variable to control which subjects see which parts of the questionnaire

Questionnaires: Using the session table

Example: Use a variable x from the subjects table.

- In the treatment create a program in the session table:

```
x = : x;
```

- In the questionnaire, use a text like:

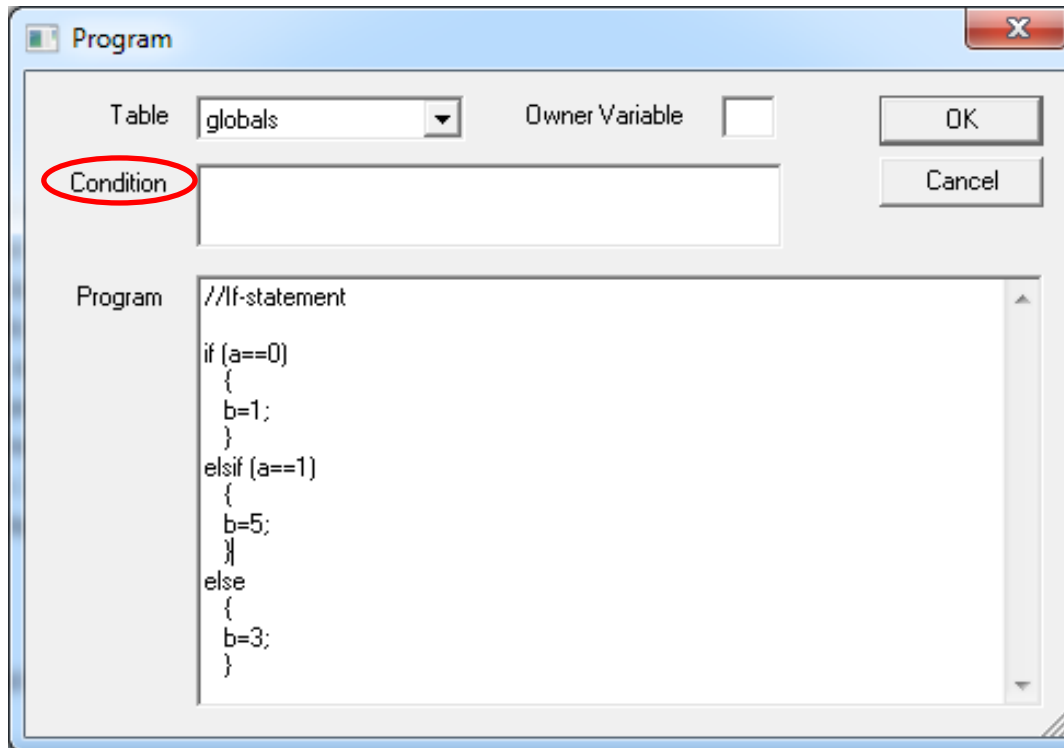
```
<> Displays x: < x | 1 >
```

Loops in z-Tree

- Loops through the records of a table:
 - `contracts.do { player = :Subject; }`
- Loops using `while ()`:
 - `while (i <= 5) { i = i + 1; }`
- Loops using `repeat { } while ()`:
 - `repeat { i = i + 1; } while (i <= 5);`
- Loops using `iterator().do`:
 - `iterator(i, 5).do { player = :Subject; }`
- Loops in time using `later () do/repeat { }`:
 - `later (15) do OR repeat { }`

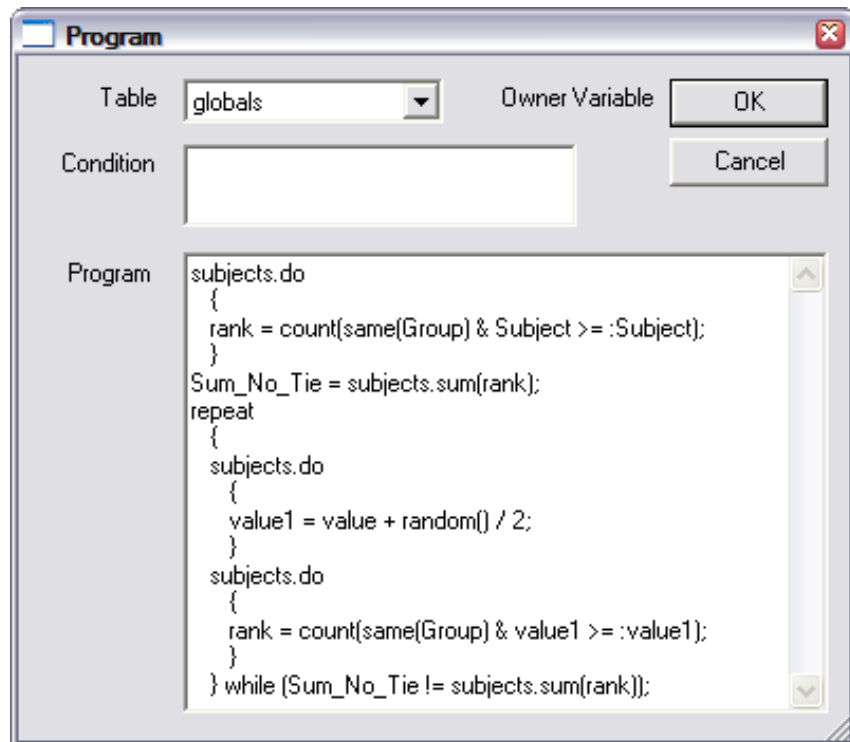
Conditional execution in z-Tree

- If-statement for value assignment:
 - `result = if (k < 5 | k >= 10, 1, 10);`
- If statement for code execution:



Calculating rank in z-Tree (from the Wiki)

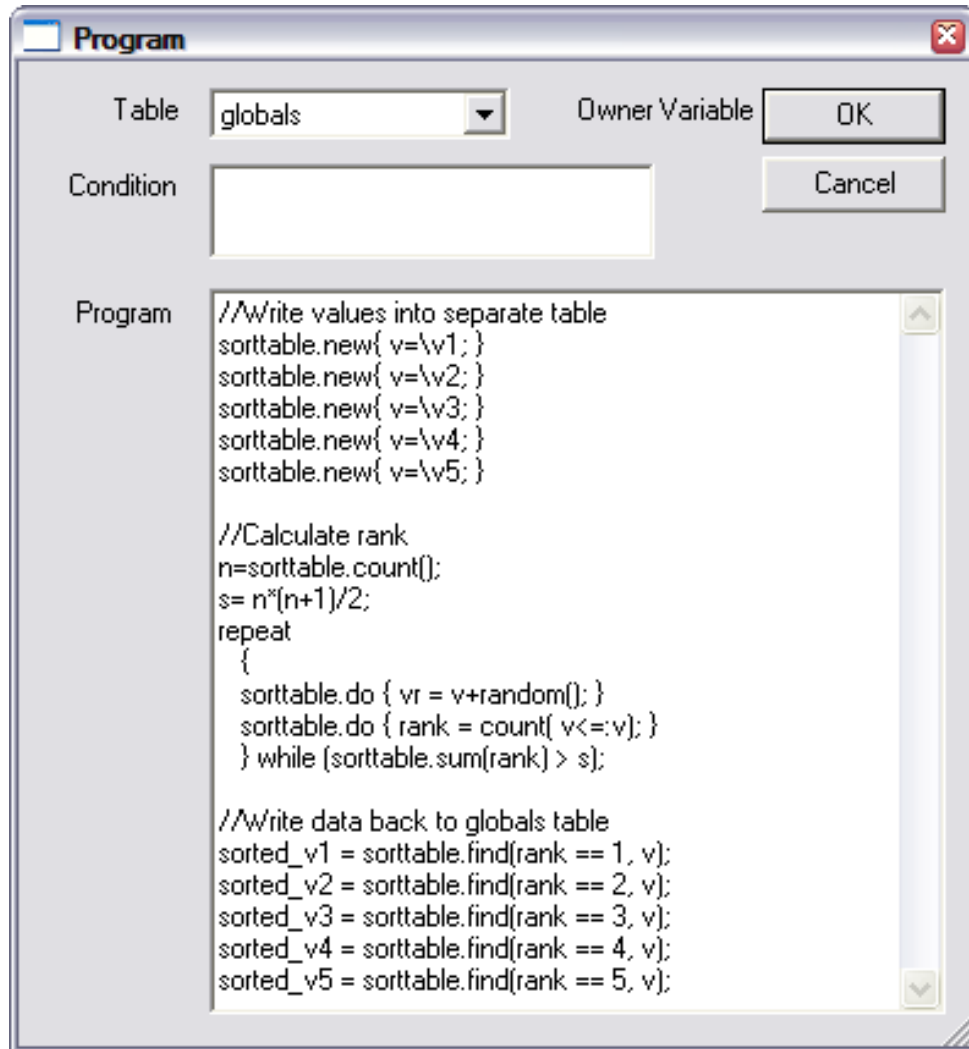
- Simple rank calculation (grouping is optional):
rank_low = count (same (Group) & value > :value) + 1;
rank_high = count (same (Group) & value >= :value);
- Rank calculation excluding ties (assuming value is integer):



```
Program
Table: globals
Owner Variable: OK
Condition:
Program:
subjects.do
{
  rank = count(same(Group) & Subject >= :Subject);
}
Sum_No_Tie = subjects.sum(rank);
repeat
{
  subjects.do
  {
    value1 = value + random() / 2;
  }
  subjects.do
  {
    rank = count(same(Group) & value1 >= :value1);
  }
} while (Sum_No_Tie != subjects.sum(rank));
```

Sorting values in z-Tree (from the Wiki)

- Assume there are integers v1 to v5 in the globals table
- Strategy:
 - Put data into a table
 - Calculate the rank
 - Transfer the data back



```
Program
Table: globals
Owner Variable: OK
Condition:
Program:
//Write values into separate table
sorttable.new{ v=\v1; }
sorttable.new{ v=\v2; }
sorttable.new{ v=\v3; }
sorttable.new{ v=\v4; }
sorttable.new{ v=\v5; }

//Calculate rank
n=sorttable.count();
s= n*(n+1)/2;
repeat
{
  sorttable.do { vr = v+random(); }
  sorttable.do { rank = count( v<=:v); }
} while (sorttable.sum(rank) > s);

//Write data back to globals table
sorted_v1 = sorttable.find(rank == 1, v);
sorted_v2 = sorttable.find(rank == 2, v);
sorted_v3 = sorttable.find(rank == 3, v);
sorted_v4 = sorttable.find(rank == 4, v);
sorted_v5 = sorttable.find(rank == 5, v);
```

Sorting values in z-Tree

- Example: Call Auction

Program in *globals* table aggregates offers in *contracts* table into price/volume list - sorted ascending in price - in *pricelist* table:

```

if (contracts.count(p>0)>0) {
  //Writes minimum and maximum price into variables in the globals table
  minprice=contracts.minimum(same(Period),p);
  maxprice=contracts.maximum(same(Period),p);
  //Sets variable price equal to the lowest price in this period
  price=minprice;
  //Loops through all prices offered in this period and writes possible purchase and sales volume
  //into pricelist table
  n=1;
  repeat {
    //Creates new entry in pricelist table
    pricelist.new {
      //Entry consists of price, volume that could be sold at this price, volume that could be bought
      //at this price, the total possible transaction volume at this price, and a counter variable
      p=:price;
      sellvol=contracts.sum(same(Period)&p<=:p&q<0,-q);
      buyvol=contracts.sum(same(Period)&p>=:p&q>0,q);
      vol=min(sellvol,buyvol);
      n=:n;}
    n=n+1;
    //Increments the price to the next offered price or to a price>maxprice if maxprice has been reached
    if (price<maxprice) {price=contracts.minimum(same(Period)&p >\price,p);} else {price=maxprice+1;} } while (price<=maxprice);
    //Sets price equal to the mean of the prices which produce maximum volume
    price=pricelist.average(same(Period)&vol==pricelist.maximum(same(Period),vol),p);}

```

Helpful advice from the manual

- Change timeout during the experiment:
 - Use a variable to set the timeout for a stage
 - Modify the variable in the period parameters in the parameter table (for periods yet to be played)
- Automatic checking of control questions
 - Use buttons with checkers in a normal treatment (i.e. not a questionnaire)

Helpful advice from the manual

- Implementing observer or experimenter subject (e.g. for dice-throw)
 - Variable that is 1 for observer and 0 for others
 - Create specific observer stage displaying relevant variables or permitting input of a variable
 - Use Participate to exclude observer from normal stages and exclude subjects from observer stage

Helpful advice

- Comment your code extensively:
- Define all variables in the Background
- Test extensively
- Attempt analysis based on data generated by test run
- Keep a documentation of tables and variables:
- Prepare session structure help sheet (“script”)

```
//Reduces liquidity and cash of old offer if it is <= liquidity of new offer
if (liquidity<=liquidity)
{
  //Switches current contract inactive, so its liquidity is not reduced in the control run
  //below
  activecontract=0;
  //Reduces cash of both subjects involved and adjusts open offers (CDA and Digital)
  subjects.do
  {
    if (Subject==:offerer | Subject==:offerer)
    {
      kapital=kapital-liquidity;
      optioncosts=optioncosts+liquidity;
      //Adjusts Digital market open offers
      redoffers.do {if (:Subject==:offerer & activecontract==1 & liquidity>:kapital) {liquidity=:kapital;}}
      //Adjusts CDA open offers
      contracts.do {if (:Subject==:buyer & seller==:offerer & preis>:kapital) {seller=-3;}}
    }
  }
  //Switches current contract active again
  activecontract=1;
  //Adjusts liquidity of both offers
  :liquidity=liquidity-liquidity;
  liquidity=0;
  activecontract=0;
}
```

contracts

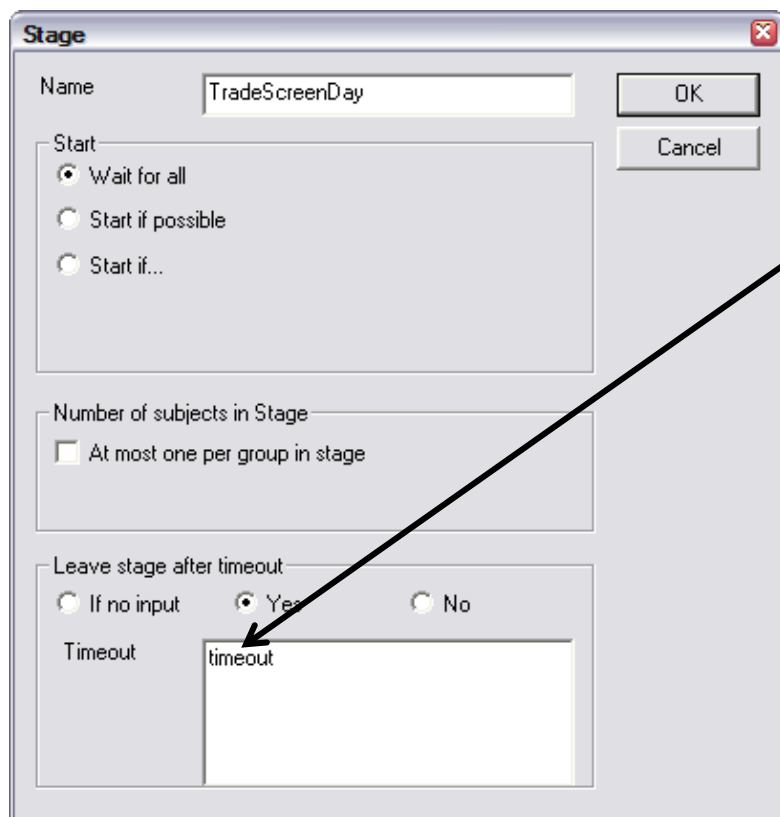
Variable	Content
Period	Period number of the record
buyer	Buying player, -1 if offered by seller and not yet accepted, -2 if deleted by control run for coverage by funds or stock, -3 if cancelled by player
creator	Creating player
preis	Sales price
seller	Selling player, -1 if offered by buyer and not yet accepted, -2 if deleted by control run for coverage by funds or stock, -3 if cancelled by player
timeaccepted	Time the contract was accepted. (+10000 during night, +20000 during WE, +30000 during day)
timecreated	Time the contract was created. (+10000 during night, +20000 during WE, +30000 during day)

stocks

Variable	Content
Period	Period number of the record
buyer	Buying player
creator	Creating player
preis	Sales price
seller	Selling player
timeaccepted	Time the contract was accepted - this is set equal to the corresponding time in the contracts table. (+10000 during WE or night)

Defining parameters in the Background

- Put general parameters into program in globals table in Background, e.g.:



Stage

Name: TradeScreenDay

Start:

- Wait for all
- Start if possible
- Start if...

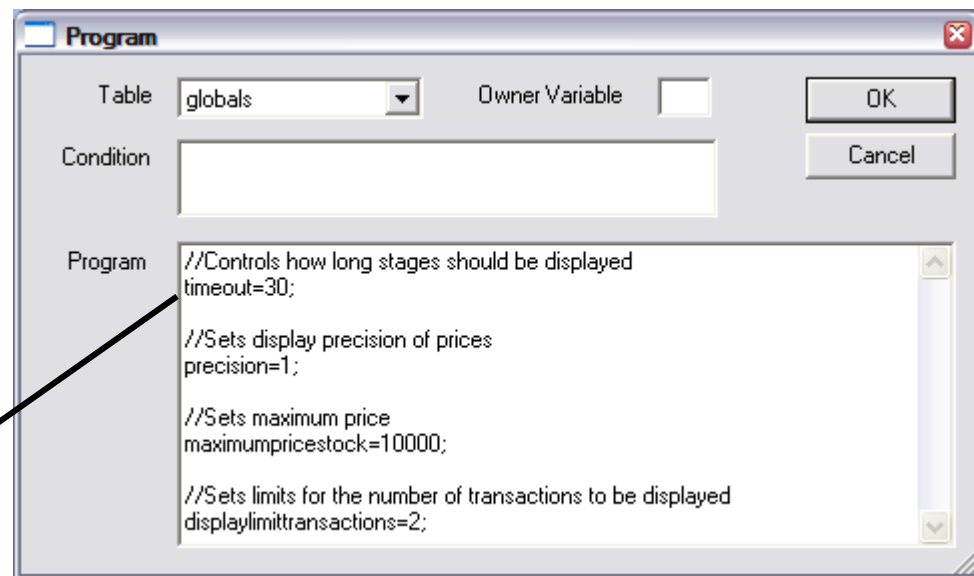
Number of subjects in Stage:

- At most one per group in stage

Leave stage after timeout:

- If no input
- Yes
- No

Timeout: timeout



Program

Table: globals

Owner Variable:

Condition:

Program:

```
//Controls how long stages should be displayed
timeout=30;

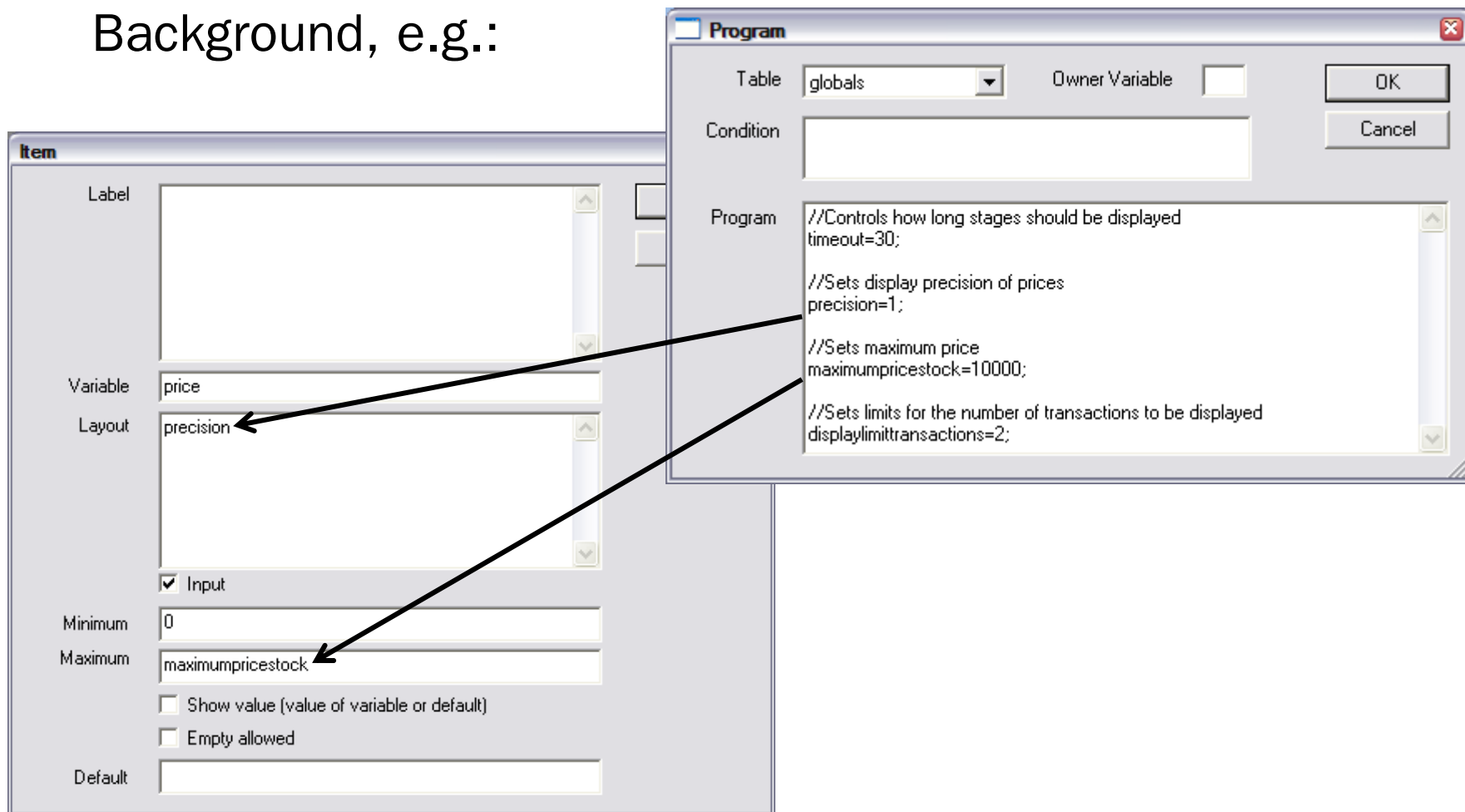
//Sets display precision of prices
precision=1;

//Sets maximum price
maximumpricestock=10000;

//Sets limits for the number of transactions to be displayed
displaylimittransactions=2;
```

Defining parameters in the Background

- Put general parameters into program in globals table in Background, e.g.:



The screenshot displays two windows from the z-Tree software. The 'Item' window on the left shows the configuration for a variable named 'price'. The 'Variable' field is set to 'price' and the 'Layout' field is set to 'precision'. The 'Input' checkbox is checked. The 'Minimum' value is 0 and the 'Maximum' value is 'maximumpricestock'. The 'Program' window on the right shows the code for the 'globals' table. The code includes comments and assignments for 'timeout=30', 'precision=1', 'maximumpricestock=10000', and 'displaylimittransactions=2'. Arrows point from the 'precision' field in the 'Item' window to the 'precision=1;' line in the 'Program' window, and from the 'maximumpricestock' field in the 'Item' window to the 'maximumpricestock=10000;' line in the 'Program' window.

Item

Label

Variable: price

Layout: precision

Input

Minimum: 0

Maximum: maximumpricestock

Show value (value of variable or default)

Empty allowed

Default

Program

Table: globals

Owner Variable

Condition

Program

```
//Controls how long stages should be displayed
timeout=30;

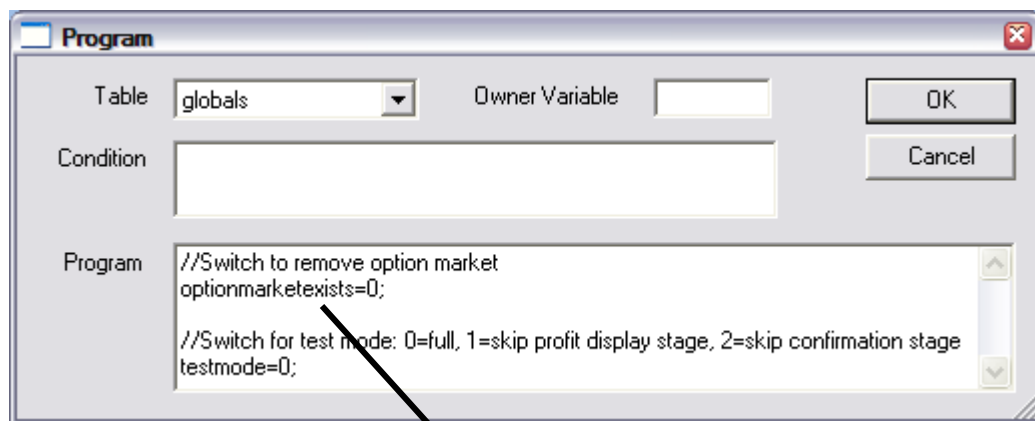
//Sets display precision of prices
precision=1;

//Sets maximum price
maximumpricestock=10000;

//Sets limits for the number of transactions to be displayed
displaylimittransactions=2;
```

Defining „switches“ in the Background

- Use variables to switch between test modes, and between different treatments within one .ztt file:



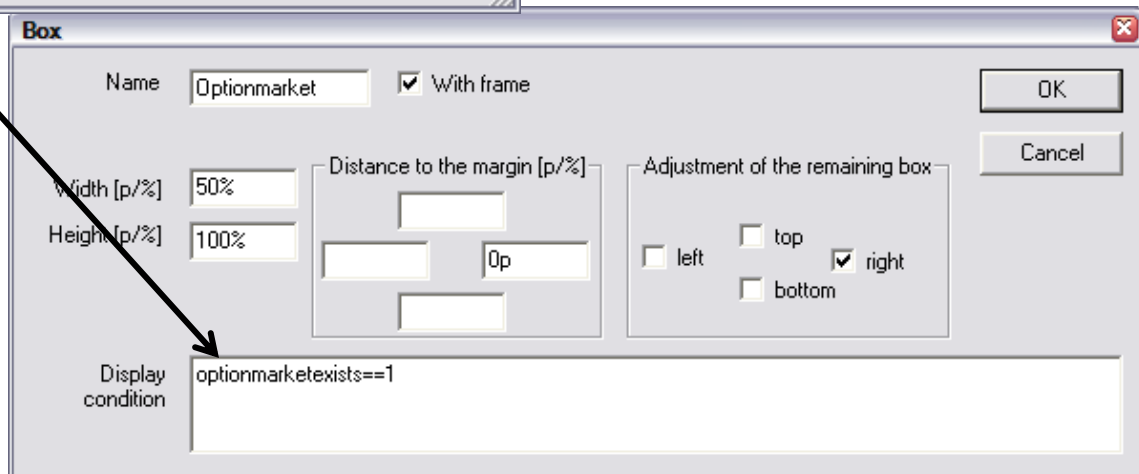
Program

Table: Owner Variable:

Condition:

Program: `//Switch to remove option market
optionmarketexists=0;

//Switch for test mode: 0=full, 1=skip profit display stage, 2=skip confirmation stage
testmode=0;`



Box

Name: With frame

Width [p/]: Distance to the margin [p/]:

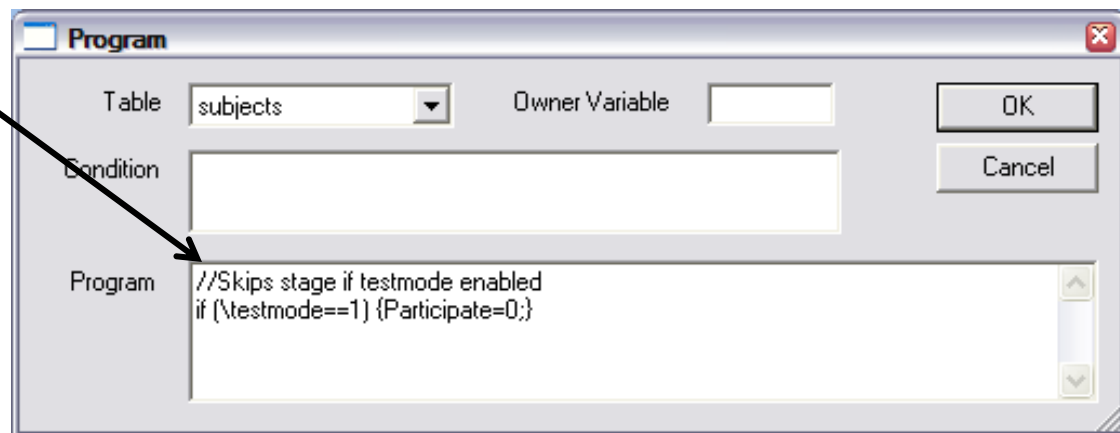
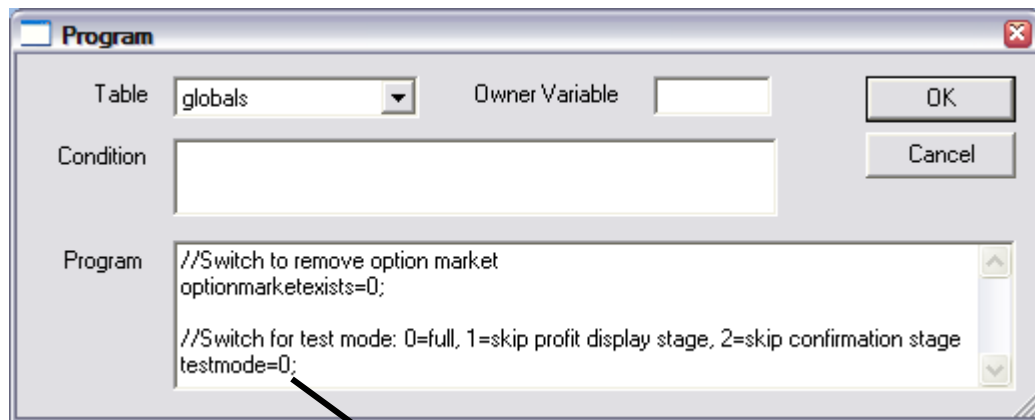
Height [p/]: Adjustment of the remaining box:

left top right bottom

Display condition:

Defining „switches“ in the Background

- Use variables to switch between test modes, and between different treatments within one .ztt file:

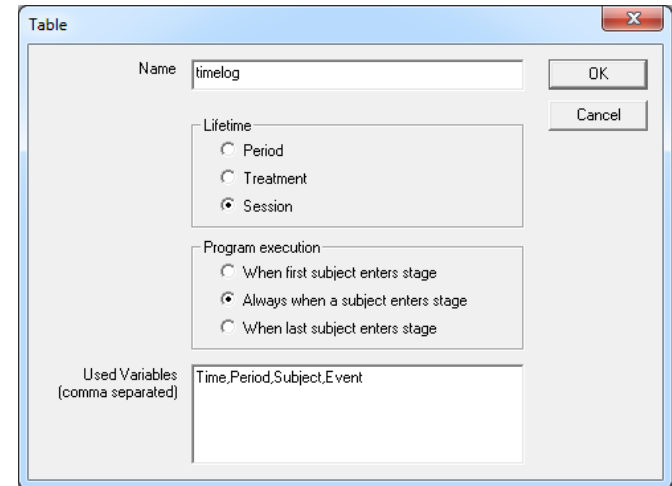


Helpful advice

- Search and replace using export function
 - Use File-Export-Treatment to export code to a .txt-file.
 - Use search/replace in Notepad/Word...
 - Re-import treatment file into z-Tree
- Calculate exact time using `gettime ()`
 - `gettime()` returns time since computer was started
 - Set variable equal to `gettime()` at beginning of period
 - Calculate difference between time of interest and time calculated at the beginning

Timelog

- Create timelog table:
 - Allows custom logging of important events
- Record starting time and create first record:



Table

Name:

Lifetime

Period

Treatment

Session

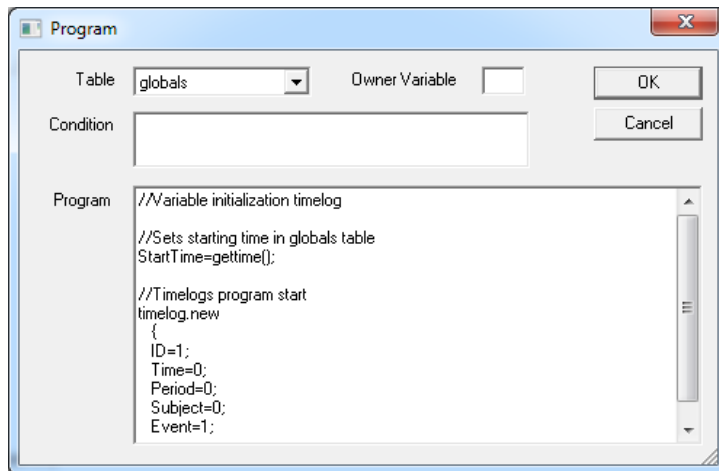
Program execution

When first subject enters stage

Always when a subject enters stage

When last subject enters stage

Used Variables (comma separated):



Program

Table:

Condition:

Program

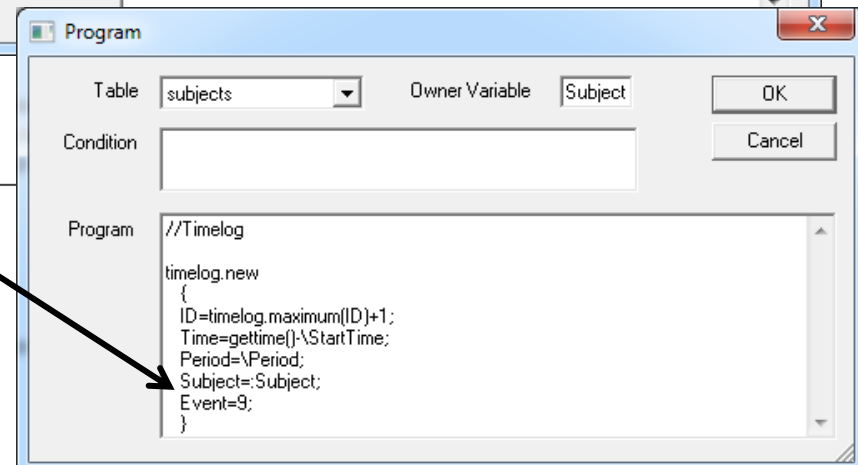
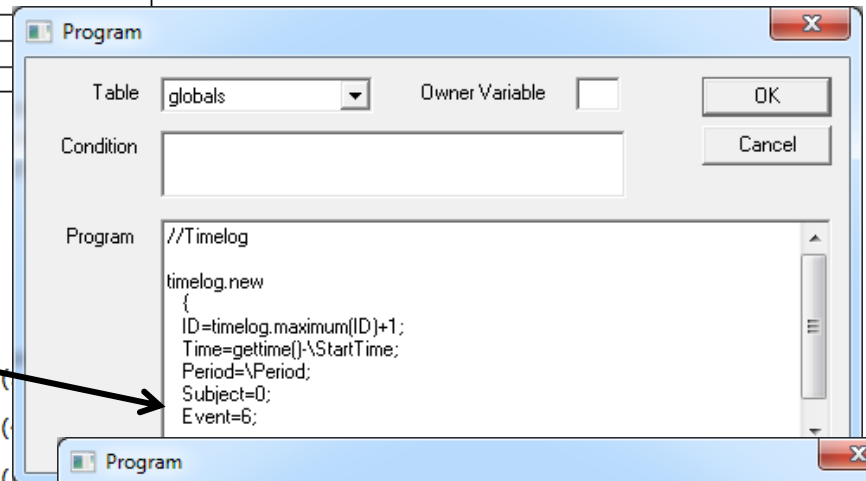
```
//Variable initialization timelog
//Sets starting time in globals table
StartTime=gettime();

//Timelogs program start
timelog.new
{
  ID=1;
  Time=0;
  Period=0;
  Subject=0;
  Event=1;
```

Timelog

Create entries for all events of interest:

Variable	Content
ID	Unique identifier of this entry
Time	Time in seconds since start of treatment
Period	Period number of the record
Subject	Subject event was triggered by, (0) if not applicable
Event	Event being logged: 1 Program start 2 Start stage Vote 3 Subject finished voting 4 Start stage VoteResult 5 Subject finished viewing VoteResult 6 Start stage Decision 7 Continue from screen Instructions(0) 8 Return to previous screen from Instructions(2) 9 Continue from screen Instructions(2) 10 Continue from screen ControlQuestion1 (3) 11 Return to previous screen from ControlQuestion1 (3) 12 Continue from screen ControlQuestion2 (4) 13 Return to previous screen from ControlQuestion2 (4) 14 Continue from screen ControlQuestion3 (5) 15 Return to previous screen from ControlQuestion3 (5) 16 Continue from screen ControlQuestion4 (6) 17 Return to previous screen from ControlQuestion4 (6) 18 Confirm decision in screen Decision (1) 19 Start stage DieRollStage 20 Start stage GeneralResults



Timelog

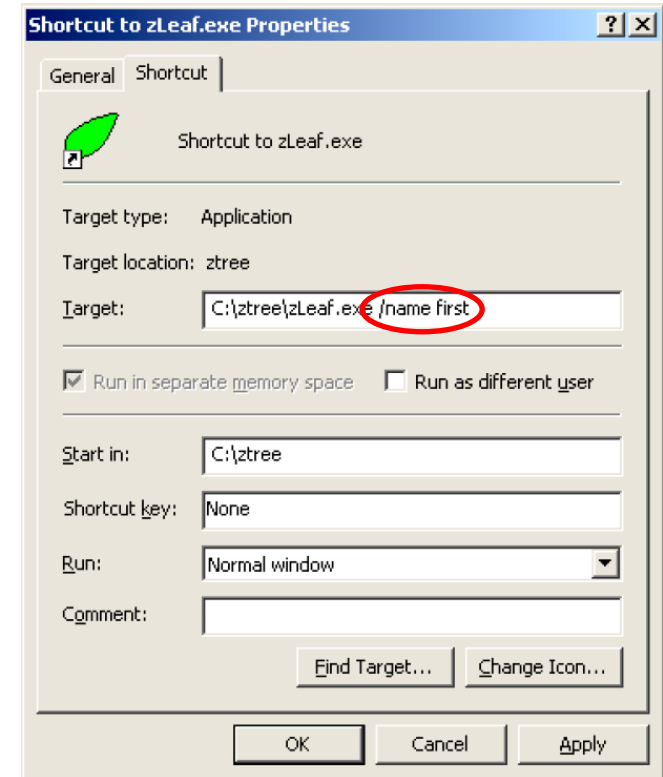
Create entries for all events of interest:

Variable	Content
ID	Unique identifier of this entry
Time	Time in seconds since start of treatment
Period	Period number of the record
Subject	Subject event was triggered by, (0) if not applicable
Event	Event being logged: 1 Program start

Period	ID	Time	Subject	Event	
0	1	0	0	0	1
1	2	0.328	0	0	2
1	3	0.421	0	0	4
1	4	0.546	0	0	6
1	5	3.916	1	1	7
1	6	6.755	1	1	8
1	7	7.457	1	1	7
1	8	7.987	1	1	9
1	9	13.588	1	1	18
1	10	16.505	2	2	7
1	11	16.833	2	2	9
1	12	22.714	2	2	18
1	13	25.257	3	3	7
1	14	25.881	3	3	9
1	15	31.746	3	3	18
1	16	31.762	0	0	19
1	17	34.133	0	0	20

Handling multiple z-leafs for testing

- Either create shortcuts:
- ...or use the following batch file:



```






leaves_variable.bat - Editor
Datei Bearbeiten Format Ansicht ?
@echo off
SETLOCAL ENABLEEXTENSIONS
SET /P leaves=How many zleafs do you want? %=%
for /L %%X IN (1,1,%leafs%) do (
start zleaf.exe /size 1024x768 /name Player%%X% /fontsize 11
PING 1.1.1.1 -n 1 -w 10 >NUL)
  
```

Notes on data processing

- Main data is saved in YYMMDD_hhmm.xls
- Reformatting possibilities:
 - Manually
 - Using z-Tree’s “Tools - Separate tables...” command (see following slides)
 - Using my Excel macro (see following slides)
 - Using Kan Takeuchi’s Stata import procedure
 - Using Oliver Kirchkamp’s R import procedure
 - (all included in the resource pack at www.palan.biz/academic - select “Downloads”)

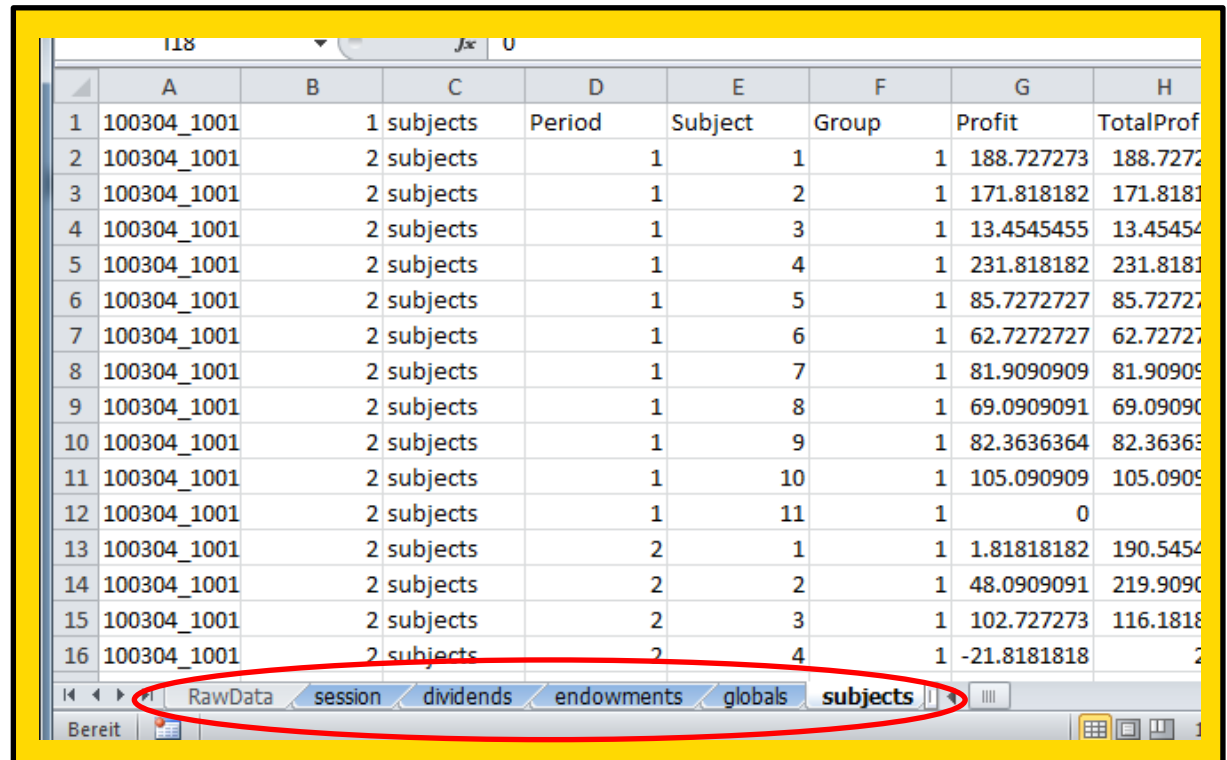
Notes on data processing

- Z-Tree’s “Tools - Separate tables...” command
 - Select .xls-file to process
 - Output:

 100928_1510.xls	28.09.2010 15:17	Microsoft Excel 97...	5 KB
 100928_1510_1_contracts.xls	14.10.2010 07:47	Microsoft Excel 97...	1 KB
 100928_1510_1_globals.xls	14.10.2010 07:47	Microsoft Excel 97...	1 KB
 100928_1510_1_parameters.xls	14.10.2010 07:47	Microsoft Excel 97...	1 KB
 100928_1510_1_session.xls	14.10.2010 07:47	Microsoft Excel 97...	1 KB
 100928_1510_1_subjects.xls	14.10.2010 07:47	Microsoft Excel 97...	2 KB
 100928_1510_1_summary.xls	14.10.2010 07:47	Microsoft Excel 97...	1 KB
 100928_1510_1_timelog.xls	14.10.2010 07:47	Microsoft Excel 97...	1 KB

Notes on data processing

- Using my Excel macro
 - Start macro
 - Answer dialog questions
 - Output:



	A	B	C	D	E	F	G	H	
1	100304_1001	1	subjects	Period	Subject	Group	Profit	TotalProf	
2	100304_1001	2	subjects		1	1	1	188.727273	188.7272
3	100304_1001	2	subjects		1	2	1	171.818182	171.8181
4	100304_1001	2	subjects		1	3	1	13.4545455	13.45454
5	100304_1001	2	subjects		1	4	1	231.818182	231.8181
6	100304_1001	2	subjects		1	5	1	85.7272727	85.72727
7	100304_1001	2	subjects		1	6	1	62.7272727	62.72727
8	100304_1001	2	subjects		1	7	1	81.9090909	81.90909
9	100304_1001	2	subjects		1	8	1	69.0909091	69.09090
10	100304_1001	2	subjects		1	9	1	82.3636364	82.36363
11	100304_1001	2	subjects		1	10	1	105.090909	105.0909
12	100304_1001	2	subjects		1	11	1	0	
13	100304_1001	2	subjects		2	1	1	1.81818182	190.5454
14	100304_1001	2	subjects		2	2	1	48.0909091	219.9090
15	100304_1001	2	subjects		2	3	1	102.727273	116.1818
16	100304_1001	2	subjects		2	4	1	-21.8181818	

Topics in data processing

- Opening .xls output file in German Excel creates problems with data interpretation (decimals interpreted as dates)
- Time data
 - Recorded for all button clicks (if stage has a timeout)
 - Times are recorded as time remaining in period:

	A	B	C	D	E	F	G	H	
1	071201_0840		1 contracts	Period	seller	buyer	creator	TimePurchaseOfferMake	Times
2	071201_0840		1 contracts		1	1	4	4	353
3	071201_0840		1 contracts		1	-3	6	6	346
4	071201_0840		1 contracts		1	8	7	7	342
5	071201_0840		1 contracts		1	-3	3	3	337
6	071201_0840		1 contracts		1	7	4	7	335
7	071201_0840		1 contracts		1	5	7	5	321
8	071201_0840		1 contracts		1	3	2	2	313
9	071201_0840		1 contracts		1	3	2	2	304
10	071201_0840		1 contracts		1	6	3	6	303
11	071201_0840		1 contracts		1	9	11	9	301

A Comprehensive Introduction to z-Tree



Stefan Palan

stefan.palan@uni-graz.at

<http://www.palan.biz/academic>