

GGIG Graphical Interface Generator

Wolfgang Britz, August 2010

- Version May 2013 -

The following report is the outcome of a collaborative effort of University Bonn and the author. Larger parts of the Java code underlying GGIG had been developed over the years in the content of the [CAPRI](#) modelling system, which receive considerably funds from the EU research framework programs. Following the general policy in CAPRI, the GGIG pre-compiled code can be used for other scientific projects as well.

The author would like to acknowledge the contribution of Alexander Gocht, vTI Braunschweig, to the CAPRI GUI coding efforts. All errors remain with the author.

Content

GGIG Graphical Interface Generator	1
Content	3
Overview	5
Current applications of GGIG	9
An overview on the GUI	9
The interface generator.....	10
Tasks.....	10
Mapping controls setting to GAMS	10
Basic concept of the control definition file	11
Worksteps.....	13
Tasks.....	13
Use of filters for exploitations.....	14
Order of the controls and layout.....	17
Fields for each definition line.....	19
Type of controls	20
Tab.....	21
Separator.....	22
Text.....	23
Checkbox.....	24
Singelist.....	25
filessel	26
Multilist / MultiListNonZero	27
Slider	30
Spinner	31
Table.....	32

Starting GAMS from GGIG	35
General interface settings	36
GAMS and R related settings	36
SVN related settings	37
Settings linked to the exploitation tools	37
Meta data handling	38
Why meta data?	38
Technical concept	38
Exploitation	40
Selecting scenarios	40
Menu bar	41
Design hints for structured programming in GAMS with GGIG	42
Using information passed from GGIG	42
Structure your program by tasks	42
One entry points for run specific settings	43

Overview

The GAMS Graphical Interface Generator (GGIG) is a tool to generate a basic Graphical User Interface (GUI) for a GAMS project with five main functionalities:

1. **Generation of user operable graphical controls from XML based definitions.** The XML file defines the project specific layout of the GUI. The user can then interact with the GUI to change the state of the controls. The state of each control component such as a checkbox can then be mapped to GAMS code (`$SETGLOALS`, Set definitions, settings for parameters). It combines hence the basic functionality of a GUI generator and a rudimentary GAMS code generator.
2. **Generation of GAMS compatible meta data** from the state of the control which can be stored in GAMS GDX format and later accessed, so that scenario definitions are automatically stored along with results.
3. **Execution of a GAMS project while passing the state of the control to GAMS** as a include file.
4. **Exploitation of results from GAMS runs** by providing an interface to define the necessary interfacing definitions in text file to load results from a GAMS into the CAPRI exploitation tools.
5. **Access to a set of GAMS related utilities.** This include e.g. a viewer for GDX files, a utility to build a HTML based documentation of the GAMS code or a batch execution utility.

GGIG is steered with text file and does not require knowledge in a higher programming language

GGIG was developed to overcome a typical problem when economic models are implemented in GAMS. GAMS itself, not at least to ensure platform portability, does not allow for graphical user input. Run specific settings for GAMS need therefore to be introduced either by changes to the GAMS project code itself or by adding settings of environment variables to the GAMS call. Experienced model users – typically the code developers themselves – know how to change run specific settings in the GAMS code, and do so typically quite efficiently. As a consequence, they seldom feel the need to invest resources in the development of a GUI steering their GAMS project. The need to invest in GUI development might have even decreased as the GAMS IDE now offers some basic functionality often found in project

specific GUIs. The IDE allows inter alia starting GAMS, inspecting parameters found in the listing file or viewing the context of a GDX file.

However, a GAMS code only solution for an economic model typically poses a serious entry barrier to newcomers for two reasons. Firstly, possible users are often not familiar with GAMS. But even with some elementary knowledge of the language, they might face problems understanding code making use of advanced GAMS features. Secondly, they face the challenge to familiarize themselves with the specific code of the project. They would need to learn enough to know exactly which specific code changes are necessary to implement e.g. scenarios in a given project. In some cases, the large and/or complex GAMS code of projects basically excludes their usage beyond some core developers. Accordingly, institutions or tool developers often observe that promising tools are only used by a few specialists, reducing returns to their investment in tool development and maintenance. Possible other users often shy away from the high learning costs and/or fear to generate, analyse and present results based on a black box where it is unclear how to enter exactly a scenario and how to access their results.

It is therefore not astonishing that some tools based on GAMS (and also on other modelling languages) have developed their specific GUIs. These GUIs let the user steer the tool with a touch & feel comparable to other programs running on modern windowed operating systems. However, writing a GUI for a larger project firstly requires considerable programming skills, often not found with the economic modellers themselves. Secondly, developing a good design, coding, debugging and maintaining a GUI can be a rather costly exercise. As a consequence, typically only rather large and well funded tools have found and invested the necessary resources to develop such GUIs. [CAPRI](#) and [runGTAP](#) provide some examples. These project specific GUIs are typically very powerful, but tend also to be tool specific. They can typically not be modified easily to fit to another GAMS project.

That renders it inviting to think about generic tools able to generate a GUI which can interface to GAMS. The coding effort can then be shared across projects, and user might even reduce learning costs if they use similar GUIs for different tools. A well-established example for such a tool is the “GAMS Simulation Environment ([GSE](#))” by Wietse Dol. GSE is quite general: it incorporates features of an Integrated Development Interface (IDE) as well as exploitation features. It is based on specific “tags” introduced in the GAMS code. GGIG is certainly not a competitor to GSE: GSE offers more functionality and is more IDE oriented. It might however be easier to embed some simple steering settings with GGIG into an existing project

compared to the tag based concept of GSE. GSE was in the past a commercial product distributed with a license but can now be downloaded for free.

An example of a completely different approach to a GUI for modelling tools offers [SEAMLESS-IF](#) with its focus on component linkage. Based on [OpenMI](#), it however requires the development of an OpenMi compatible wrapper around the GAMS project itself.

Concepts such as the SEAMLESS-IF are therefore probably only suitable for larger projects focusing on combining components based on different programming languages. SEAMLESS-IF is further on based on a client/server implementation and requires specific software licences for deployment.

GGIG might hence be seen as a quite simple and easy to use tool to generate GUIs for GAMS projects. If all GGIG features are used, it can however host quite complex projects. The new GUI for CAPRI built with GGIG offers an example for a rather complex implementation.

As mentioned above, a second important contribution of GGIG is to mechanize to the largest extent the generation, storage and later inspection of meta data underlying a scenario and the related result set, overcoming an often encountered weakness in (economic) models.

And thirdly, GGIG offers a bridge between the powerful [CAPRI exploitation tools](#) and other GAMS based models. It draws on the experiences with [BenImpact](#), [MIVAD](#) and the village CGEs developed in [Advanced-Eval](#), GAMS tools models resp. Java based GUIs where the CAPRI exploitation tools had been used. These GAMS based projects used the CAPRI exploitations, but did not add any GUI functionalities to also steer their models. The experiences with these examples can hence be seen as the starting point for the development of GGIG in order to expand beyond a pure, project adjusted implementation of the CAPRI exploitation tools.

Some specific skills and eventually serious refactoring of the reporting part of an existing model are necessary to benefit from the full functionality of the CAPRI exploitation tool. It therefore pays typically off to start using GGIG for exploitation from the beginning. But at least, no skills in coding in a higher programming language such as Java are necessary to define the necessary interfaces between the GAMS project and the exploitation part. The latter offers interlinked tables (with selections, sorting, outlier control, pivoting), different type of graphs, maps and flow maps.

Additionally, GGIG features a set of utilities originally developed for CAPRI such as HTML based documentation of the GAMS code or a GDX viewer.

The development of GGIG would have been impossible without the continued funding for the maintenance and development of CAPRI by the EU Commission, which also led to the emergence of the CAPRI GUI and exploitation tools. That code base was the starting point for GGIG. I would also like to mention the contribution of Alexander Gocht over the last years to the code of the interface.

The main parts of GGIG are graphically depicted below. At its core stands the GGIG Control generator, based on Java code. Based on a XML based definition file, it generates a project specific GUI which can be operated by the user. The state of these controls such as numerical settings, on/off settings or n of m selection can be passed to GAMS in automatically generated include file, including automatically generated meta data. The user can also execute GAMS from the GUI. The GUI can equally load numerical results and meta data in a specific GDX viewer. The latter supports “view definition”, i.e. pre-defined reports to exploit the results. The details of the different elements are discussed below.

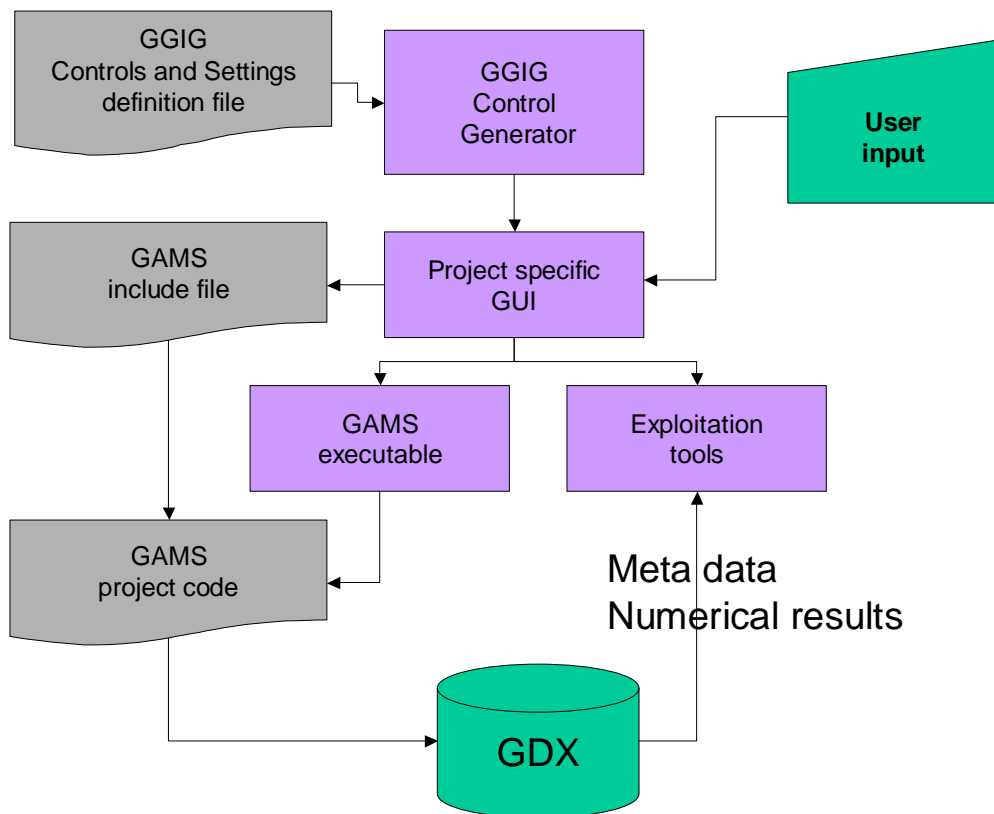


Diagram: Overview on information flow in GGIG

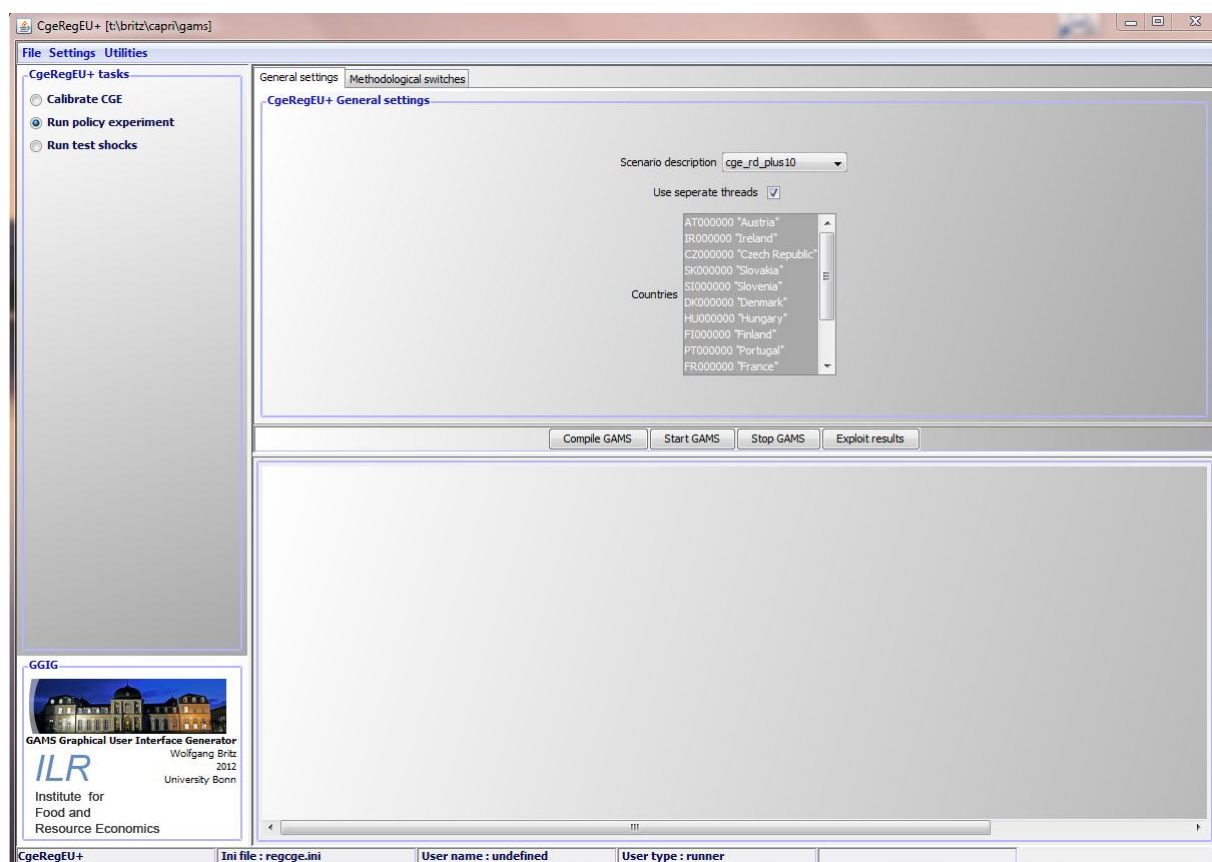
Current applications of GGIG

Since the first prototype, GGIG has been successfully implemented in a number of projects:

- [DairyDyn](#): an fully dynamic single farm model focusing on the impact of Green House Gas emission indicators on allocation and investment decision
- A [small, spatial multi-commodity model](#) for world trade of cooked and uncooked poultry meat with a focus on trade bans related to Avian Influenza
- A EU wide layer of regional CGEs with a focus on Rural Development measures on the second pillar of the CAP
- [LANA-HERBAMO](#): A Hydro-Economic model for the lake Naivasha in Kenya.

These projects has helped to clarify some requirement and triggered the implementation of new and expanded features. Since 2013, CAPRI uses GGIG. Further applications are already underway, e.g. in the context of the [FADNTOOL](#) project.

An overview on the GUI



As shown above, the GUI consists a few elements:

1. A **menu bar** which allows to change some settings (see the section on general interface settings)
2. A **workstep** and **task selection panel** on the left hand side where the user can select between different tasks belonging to the project.
3. A **right hand side panel** which either shows:
 - i. The generated controls, a button panel to start GAMS and a windows in which the message log from GAMS is shown
 - ii. A panel to select scenario and to start the exploitation
 - iii. The exploitation tools
4. A small window in the left lower corner which present a logo.

Whereas the elements 1. and 3.ii and 3iii. are not project specific, the worksteps and tasks available in 2. and the controls shown to the user in 3.i. are generated in a project specific initialisation file. The details of that file – which is core of GGIG – are discussed below.

The interface generator

Tasks

Tasks are central elements in GGIG. Each control can belong to one or several task, and each task might have its own GAMS process. That allows steering even rather complex GAMS installations with one GUI. It allows supports a structured development of the GAMS code as either separate GAMS files with a clear purpose are generated or a GAMS file consists of blocks which belong to certain tasks.

When the user selects a task, only the controls belonging to that task are shown to the user, easing the handling of the GUI.

Mapping controls setting to GAMS

Controls are user operable, graphical elements. A few examples are shown below.





Diagram: Example of controls generated with GGIG

In the case of GGIG, the controls are used by the user to define textual and numerical settings which in turn define run specific settings for a GAMS project. GGIG offers five functionalities related to these controls and their interactions with a GAMS project:

1. It *generates the controls* from a definition file on a windowed program interface.
2. It offers the necessary code to *intercept user operations* on the controls.
3. It maps the settings of the controls based on the user input to as sequence of *GAMS statements*, which can be included into a GAMS project to generate a specific run.
4. It allows *execution of GAMS*.
5. It offers a GDX viewer which supports the definition of *pre-defined reports*.

The overview on the process is shown in the diagram above.

In order to allow the run specific settings to enter a specific GAMS project, the generated include file should define the sole entry point of run specific information. The state of the controls – passed to the include file - should hence define all the necessary information for a specific run. The GAMS code should accordingly not allow for or require additional changes to generate a “scenario”, i.e. a specific run. It is however easily to use a text control to enter the directly the name of a include file.

The generated include file is overwritten each time the user starts the GAMS project.

Basic concept of the control definition file

GGIG support two format for definitions file: XML based property files or standard Java property files. The later are only supported for backward compatibility and should no longer be used for new GGIG projects.

XML property file

The core of GGIG consists of the control definition file. The XML property file allows breaking up the settings for a control, task etc. into several XML tags, and these tags can additional by stored in different lines, see example below:

```
<control>
  <order>1110</order>
  <Type>singlelist</Type>
  <Title>First year</Title>
  <Value>1984</Value>
  <Options>1984,1985,1986,1987,1988,1989,1990,1991,1992,1993,1994,1995,1996,</Options>
  <gamsName>FirstYear</gamsName>
  <tasks>Prepare national database,
  Finish national database,
  FSS selection routine,
  Build regional time series,
  Build regional database,
  Build global database,
  Generate trend projection,
  Generate farm type trends,
</tasks>
</control>
```

The keywords are discussed in detail below.

Standard Java property files

It follows the basic implementation of a property file in Java. Each line thus consists of a key – value pair, separated by an equal sign. The definition of the controls is stored in the same file along with general settings such as the name of the GAMS project, directories, the user name etc..

For each control, one line is used. That line comprises all the necessary information to generate the control, as well as to store the current setting.

The control definition file is text based and can hence be edited with any text editor. Most of the settings – with the exemption of the definitions of the controls themselves – can also be entered by the user via the controls on the GGIG interface. These project independent controls are to a larger extent borrowed from the CAPRI user interface. On top, a first rudimentary control editor is embedded in the tool.

Call of GGIG

In a normal installation there are two ini files:

1. One default file with the control definitions and related default values. That file should be typically under version control. It can be in XML or ini file format.
2. A second file which is installation specific, it will store the values entered by the user and will be in the ini format

A typical call will therefore look like:

```
Gig.jar project.ini project_default.ini or Gig.jar project.ini project_default.xml
```

It is hence possible to host several GGIG based installations in one directory where the jars etc., are stored.

Worksteps

Worksteps allow to group tasks. The following attributes are possible

Name Name of the workstep shown as selectable radio button (required)

Tasks List of tasks

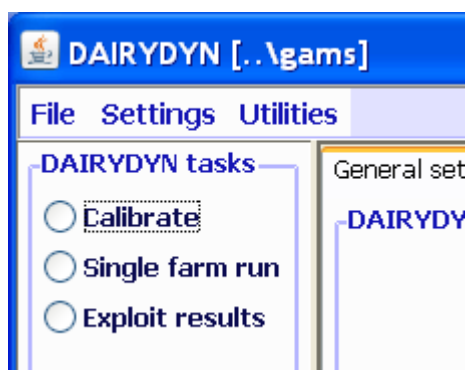
```
<workstep>
  <name>Build database</name>
  <tasks>Prepare national database,
          Finish national database,
          FSS selection routine,
          Build regional time series,
          Build regional database,
          Build global database,
          Build HSMU database
        </tasks>
</workstep>
```

Tasks

The control definition file defines a list of task (such as calibrating the model and running the model). A task can have its own GAMS file to start, its own result directories and its own set of controls. Each control can be shared by several tasks.

```
<task>
  <name>Run test shocks with CGE</name>
  <gamsFile>regcge.gms</gamsFile>
  <incFile>regcge_settings</incFile>
  <regionDim>0</regionDim>
  <Dim5Dim>1</Dim5Dim>
  <activityDim>2</activityDim>
  <productDim>3</productDim>
  <scenDim>4</scenDim>
  <yearDim>5</yearDim>
  <useMeta>true</useMeta>
  <resdir>regcge</resdir>
  <filemask>res_[0-9]{4}testShocks.{1}gdx$</filemask>
</task>
```

The tasks are put on the interface in alphabetical order:



The following attributes are possible for a task

Name	defines the name of task, shown on interface (required)
gamsFile	defines the name of the GAMS project to start (optional)
resDir	result directory where the results are stored (optional)
filemask	regex string used filter the files shown in the scenario exploitation boxes for the task
incFile	defines the name of include file used by the task (optional)
gdxsymbol	defines the GAMS symbol (set,parameter) to load for exploitation
{logical}dim	position of the logical dim in gdxsymbol, where logical=region, activity, product, year, scen, dim5
filters	filters for scenario input, see below

If no *gamsFile* or *resDir* are given, the general ones defined in the ini-file are used.

Use of filters for exploitations

Filters are used to

1. To let the user select from the GDX files which are potentially generated by the task based on a specific content selection, .e.g. only files from a specific year
2. To introduce a filter on the GDX element loaded in the viewer, e.g. to only load records for a specific country

A filter definition consist of 3 or 4 fields:

1. The logical dimension to which it is applied: {region, activity, product, year, scen, dim5}

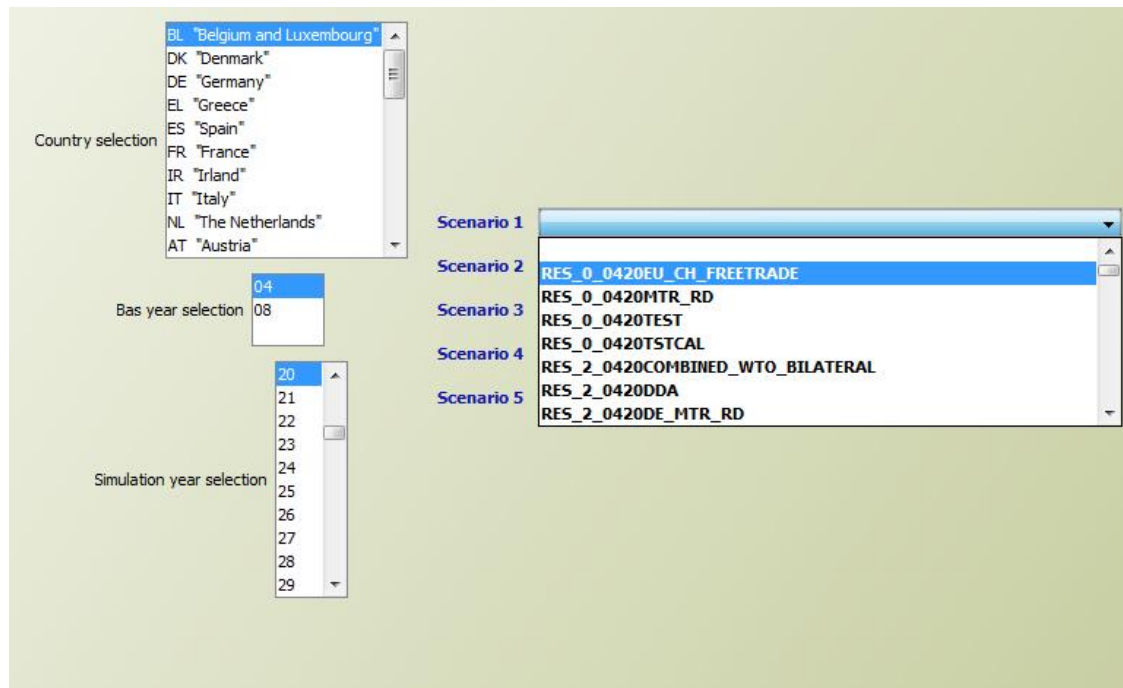
2. The selection control which is used for the filter
3. The type of filter:
 - a. **“Starts_with” or “ends_with” for GDX element filters**, i.e. only such records will be loaded where the item describing the logical dimension starts with one of the selected keys.
 - b. Otherwise, a pair of integer values which describe on which position of the file names the selected key should be found plus either “skip” for only using selecting files or “merge” to merge records from the chosen GDXs.

The screenshot below shows an example with the following filters:

```
<filter>region,CountriesSel,starts_with</filter>  
<filter>Base_year,BaseYearsSel,7,8,skip</filter>  
<filter>Year,SimYearsSel,9,10,skip</filter>
```

The first filter “starts_with” does not affect the file selection, but will affect the records loaded in the viewer. In our example, only records where the region key starts with “DE” or “BL” will be available.

The other two filters will skip files where the base and simulation years do not match the selection. In our example, the base year is stored as a two digit key on position 7 and 8, and only files with a “04” are in the drop down box for the scenarios. Similarly, only results for the simulation year “20” are selected.



Normally, the name of the file will be used to characterize the “scenario”. The “merge” is made for the case where several GDX files should be combined and the file name does not distinguish model runs. An example offers the downscaling component of CAPRI: it produces in separate GAMS run for the same scenario one file for each country which comprise rather huge data sets. The “merge” mode allows combining these result sets together.

Order of the controls and layout

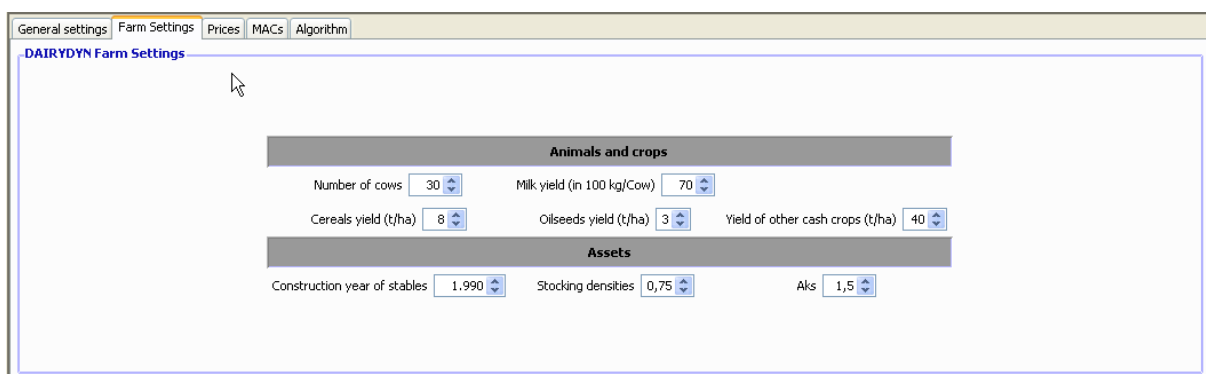
The controls will be placed in alphabetical sorted order of their keys on the interface.

Normally, each control copies one line. If the keys for a range of controls end with consecutive numbers (such A110, A111, A112 ...), the controls are put in the same line.

The following definition statements

```
C100=Type\tab;Title\=General settings
C105=Type\text;Title\=Scenario description;Value\=30Cows
C110=Type\slider;Title\=Last year;Value\=2020;range\=2015,2100,1,15;gansName\=lastYear
C120=Type\spinner;Title\=Time resolution for investment/off farm labour decisions;Value\=2.0;range\=1,10,1;gansName\=timeResolution
C130=Type\spinner;Title\=Max yearly growth rate of cow herd (%);Value\=4;range\=0.0,10,.0.5;gansName\=maxGrowthRateCowherd
C140=Type\checkbox;Title\=Allow for reduction of max milk yield;Value\=true;gansName\=mlkRed
C200=Type\tab;Title\=Farm Settings
C210=Type\separator;Title\=Animals and crops
C220=Type\spinner;Title\=Number of cows;Value\=30.0;range\=0,120,5;gansName\=nCows
C221=Type\spinner;Title\=Milk yield (in 100 kg/Cow);Value\=70.0;range\=40,100,1;gansName\=milkYield
C223=Type\spinner;Title\=Cereals yield (t/ha);Value\=8;range\=4,10,1;gansName\=cereYield
C224=Type\spinner;Title\=Oilseeds yield (t/ha);Value\=3;range\=2,4,0.2;gansName\=oilsYield
C225=Type\spinner;Title\=Yield of other cash crops (t/ha);Value\=40;range\=30,40,2;gansName\=restYield
C230=Type\separator;Title\=Assets
C240=Type\spinner;Title\=Construction year of stables;Value\=1990.0;range\=1980,2010,5;gansName\=stableYear
C241=Type\spinner;Title\=Stocking densities;Value\=0.75;range\=0.5,2.0,0.1;gansName\=stockingDens
C242=Type\spinner;Title\=Aks;Value\=1.5;range\=0.5,3.0,0.5;gansName\=Aks
```

will hence show on the interface as seen in the screen shots below. All controls following a “tab” control will be put on that tab (until the next one).



With the XML-based definition, the order field is used.

```
<control>  
  <order>1120</order>  
  <Type>singlelist</Type>  
  <Title>Base year</Title>  
  <Value>2004</Value>  
  <Options>2004,2006,2008</Options>  
  <gamsName>BaseYear</gamsName>  
  <tasks>Prepare national database,Build regional time series  
</control>
```

Fields for each definition line

The necessary information to control is stored in a line of the control definition file. The following fields are available:

Type	defines type of control (required). The different types are discussed below in detail.
Title	defines description of control as seen by user (required)
GamsName	defines names of global settings resp. SET name (optional)
Value	pre-selected setting (optional)
Options	list of available options (required where applicable)
Range	Min, max, increment, major ticks; or number of rows shown (required where applicable)
Tasks	List of tasks to which the control belongs. If empty, it belongs to all tasks
Tooltip	A tooltip text hovering over the control
Pdflink	Link to a pdf file and chapter to open on mouse over
Selgroups	Selection list opened by pop-up menu (see Multilist control)
Disable	Control is blocked for input – useful to show settings on interface which are should be sent to GAMS for a specific task.

Type of controls

The following types of controls are available. The related JAVA swing JComponent is shown in bracket.

Tab	Introduces a new tab on the tabbed plane hosting the controls
Separator	to structure a pane with control (JLabel in an JPanel with a border)
Panel	the next controls are shown together on a panel
Text	to enter a free text (JTextField)
Checkbox	for on-off type of settings (JCheckBox)
Singlelist	for 1 of n selections (JList in a JScrollPane)
RadioButtons	for 1 of n selections (Group in JButton, vertically aligned)
Filesel	for 1 of n selections of a list of files (JList in a JScrollPane)
Multilist	for n of m selections (n=0..m), (non editable JComboBox)
MultilistNonZero	for n of m selections (n=1..m), (non editable JComboBox)
Slider	for integer value selection from a range of values (JSlider)
Spinner	for floating or integer value selection from a range of values (JSpinner)
Table	to enter floating point variables in a two-dimension parameter (JTable)

Tab

Purpose

Used to structure the interface by grouping controls on an input pane: introduces a new tabbed plane to which controls following are then added

Applicable fields:

Title, Tasks

Control optic:



Remark:

The user can only see one of the tab pane at any time – care should hence be given to keep the number of tabs and the assignment of controls to tabs such that a user can easily check all key inputs.

Separator

Purpose

Used to structure the interface, gives a title for the next block of controls

Applicable fields:

Title, Value, Tasks

Control optic:



Settings for objective function

Example definition:

```
<control>
  <order>2010</order>
  <Type>Separator</Type>
  <Title>Supply model</Title>
  <tasks>Baseline calibration market model,
  Baseline calibration supply models,
  Run scenario with market model,Generate expost results
  Run scenario without market model</tasks>
</control>
```

Text

Purpose

To enter text. Typically used to parse the name of the scenario to GAMS

Applicable fields:

Title, Value, Tasks

Control optic:

Scenario description

Possible value:

Any text allowed

User action:

Edit with keyboard

Example definition:

```
<control>
  <order>1450</order>
  <Type>text</Type>
  <Title>Alternative GAMS license file for GHG emission estimation</Title>
  <Value>gamslice_cplex</Value>
  <Options> </Options>
  <range>0</range>
  <gamsName>altLicense</gamsName>
  <tasks>Baseline calibration market model,Run scenario with market model,Generate expost results</tasks>
</control>
```

Output to GAMS:

```
$SETGLOBAL Scenario_description my first scenario
```

Checkbox

Purpose

Used for on/off settings, i.e. in cases where one of two options must be chosen, e.g. in cases of project modules which can be used or not (1 of 2). Should not be used for 1 of n selections where $n > 2$ – use a List instead.

Applicable fields:

Title, GamsName, Value, Tasks

Control optic:

Use branching priorities

Possible value:

true, false

User action:

tick / untick box with mouse

Example definition:

```
<control>
  <order>1020</order>
  <Type>CheckBox</Type>
  <Title>Generate GAMS child processes on different threads</Title>
  <Value>true</Value>
  <gamsName>threads</gamsName>
  <tasks>
    Build HSMU database,
    Run scenario with market model,Generate expost results
    Run scenario without market model,
    Baseline calibration supply models,
    HSMU baseline,
    Downscale scenario results,
  </tasks>
</control>
```

Output to GAMS:

```
.$SETGLOBAL Priorities false
```


Singelist

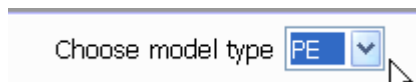
Purpose

Used for 1 of n selections. Used in cases where more than 2 mutually exclusive values for a setting are available.

Applicable fields:

Title, GamsName, Value, Options, Tasks

Control optic:



Note: Drop down list will appear if the user clicks on arrow.

Possible value:

Defined by options field

User action:

tick / untick one of the selection possibilities with mouse

Example definition:

```
<control>
  <order>1110</order>
  <Type>singlelist</Type>
  <Title>First year</Title>
  <Value>1984</Value>
  <Options>1984,1985,1986,1987,1988,1989,1990,1991,1992,1993,1994,1995,1996,</Options>
  <gamsName>FirstYear</gamsName>
  <tasks>Prepare national database,
  Finish national database,
  FSS selection routine,
  Build regional time series,
  Build regional database,
  Build global database,
  Generate trend projection,
  Generate farm type trends,
</tasks>
</control>
```

Output to GAMS:

```
$SETGLOBAL Choose_model_type PE
```

filessel

Purpose

Used for 1 of n selections of a list of files. That is e.g. interesting when the user can chose from a list of pre-existing scenario definitions in GAMS files.

Applicable fields:

Title, GamsName, Value, Options, Tasks

Control optic:

Scenario description

Note: Drop down list will appear if the user clicks on arrow.

Possible value:

*Defined by the file selection string in options field, .e.g
..\gams\pol_input\cge_*.gms. The file extension will be automatically removed from the items.*

User action:

tick / untick one of the selection possibilities with mouse

Example definition:

```
<control>
  <order>1010</order>
  <Type>filessel</Type>
  <Title>Scenario description</Title>
  <Value>MTR_RD</Value>
  <Options>..\gams\pol_input\*.gms</Options>
  <range>0</range>
  <gamsName>result_type</gamsName>
  <tasks>Baseline calibration market model,
    Baseline calibration supply models,
    HSMU baseline,
    Run scenario with market model,Generate expost results
    Run scenario without market model,
    Run scenario only with market model
    Downscale scenario results
  </tasks>
  <tooltip>Name of the scenario file to run. The results will be stored under the name as well.</tooltip>
</control>
```

Output to GAMS:

```
$SETGLOBAL scenDes cge_rd_noChg
```

Multilist / MultiListNonZero

Purpose

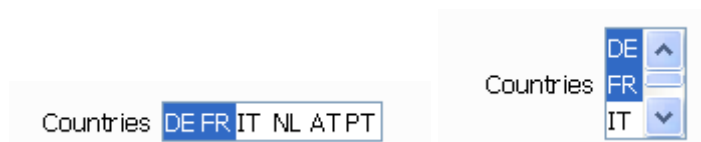
Used for m of n selections, i.e. in cases where features are not mutually exclusive.

Multilist allows $m = 0$, i.e. also empty selection. MultiListNonZero requires $m > 0$, i.e. at least one element must be selected.

Applicable fields:

Title, GamsName, Value, Options, range, Tasks

Control optic:



Notes:

- left hand side: range=0 right hand side: range = 3
- Drop down list will appear if the user clicks on arrow, and number of elements > range and range<>0

Possible value:

Defined by options field

User action:

tick / untick box fields in the control with mouse

Example definition:

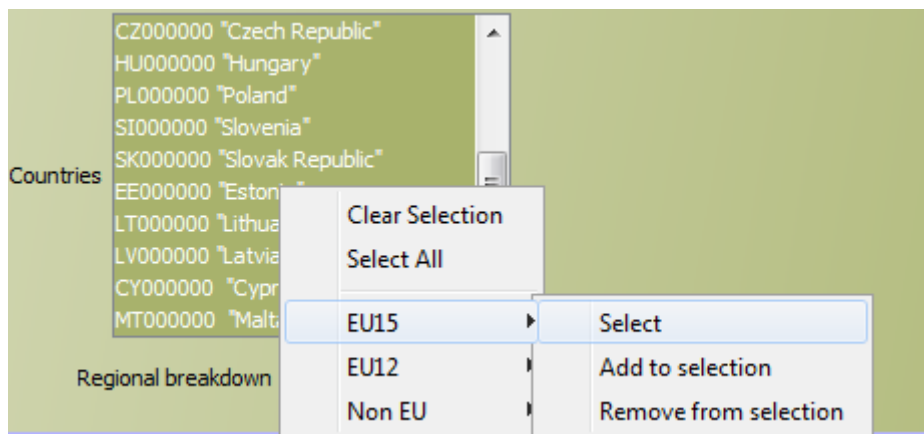
```
<control>
  <order>1426</order>
  <Type>multilist</Type>
  <Title>Longrun Option</Title>
  <Options>FA02050 "Fao projections"
    GLOBIOM_EU "Projections with GLOBIO EU model"
    GLOBIOM_GL "Projections with global GLOBIOM model"
  </Options>
  <Value>FA02050 "Fao projections"</Value>
  <range>3</range>
  <gamsName>longrunScen</gamsName>
  <tasks>Build global database</tasks>
</control>
```

Output to GAMS:

```
SET Countries /  
DE  
FR  
/;
```

Selection groups

The *multilist* control features a pop-up menu which without selection groups only allows to clear the selection or to select all items (see below).



Selection groups can be added which allow for groups of items to select them or add respectively remove from the selection. Each selection group starts with a forward slash “/” following by the name of the group. The items are and the next selection group are then comma separated as shown below. Commas can be skipped if the next item is on a different line.

```
<selGroups>
  /EU15
    BL000000 "Belgium and Luxembourg",
    DK000000 "Denmark",
    DE000000 "Germany",
    EL000000 "Greece"
    ES000000 "Spain"
    FR000000 "France"
    IR000000 "Ireland"
    IT000000 "Italy"
    NL000000 "The Netherlands"
    AT000000 "Austria"
    PT000000 "Portugal"
    SE000000 "Sweden"
    FI000000 "Finland"
    UK000000 "United Kingdom"
  /EU12
    CZ000000 "Czech Republic"
    HU000000 "Hungary"
    PL000000 "Poland"
    SI000000 "Slovenia"
    SK000000 "Slovak Republic"
    EE000000 "Estonia"
    LT000000 "Lithuania"
    LU000000 "Latvia"
    CY000000 "Cyprus"
    MT000000 "Malta"
    BG000000 "Bulgaria"
    RO000000 "Romania"
  /Non EU
    NO000000 "Norway"
    TUR "Turkey"
    AL000000 "Albania"
    MK000000 "Macedonia"
    CS000000 "Serbia"
    MO000000 "Montenegro"
    HR000000 "Croatia"
    BA000000 "Bosnia and Herzegovina"
    KO000000 "Kosovo"
</selGroups>
```

Slider

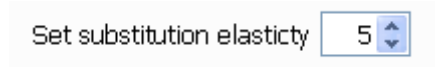
Purpose

Used to selected one integer value from a given range of allowed ones. The increments must also be defined.

Applicable fields:

Title, GamsName, Value, Options, range, Tasks

Control optic:



Note: Selectable values will be restricted according to the increment definition.

Possible value:

Defined by range field

User action:

Select value by pressing up/down arrows or by editing the field with keyboard

Example definition:

Type=spinner;title=Set substitution elasticity;range=0,10,0.5;value=5

Output to GAMS:

```
.$SETGLOBAL Set_substitution_elasticity 5
```

Spinner

Purpose

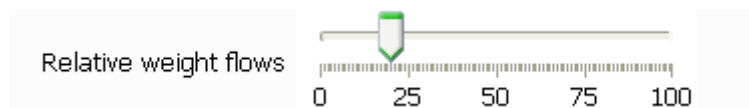
Used to selected a integer value from a range of allowed ones . The increment is always unity. Could be internally used as a floating value, e.g. by using it for shares in percentages terms.

If the range of the spinner is large, it might be hard for the user to pick a specific value. In that case, a spinner is easier to control.

Applicable fields:

Title, GamsName, Value, Options, range, Tasks

Control optic:



Possible value:

Defined by range field

User action:

Select value by moving slider

Example definition:

```
<control>
  <order>1011</order>
  <Type>spinner</Type>
  <Title>min end year of planning horizon</Title>
  <Value>2020</Value>
  <range>2015,2100,1,15</range>
  <gamsName>lastYearMin</gamsName>
  <tasks>Experiments</tasks>
</control>
```

Output to GAMS:

```
$SETGLOBAL Relative_weight_flows 20
```

Table

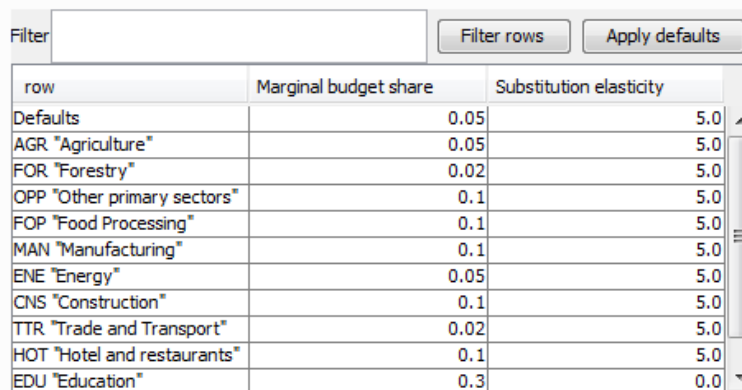
Purpose

Define a table with floating point values passed to GAMS.

Applicable fields:

Title, GamsName, Value, Columns, Rows, Dim3s, Range, Tasks

Control optic:



The screenshot shows a table with a filter input field at the top left, containing the text "Filter". To the right of the filter are two buttons: "Filter rows" and "Apply defaults". The table itself has three columns: "row", "Marginal budget share", and "Substitution elasticity". The rows are: Defaults, AGR "Agriculture", FOR "Forestry", OPP "Other primary sectors", FOP "Food Processing", MAN "Manufacturing", ENE "Energy", CNS "Construction", TTR "Trade and Transport", HOT "Hotel and restaurants", and EDU "Education". The values for "Marginal budget share" range from 0.02 to 0.3, and for "Substitution elasticity" from 0.0 to 5.0. A vertical scrollbar is visible on the right side of the table.

row	Marginal budget share	Substitution elasticity
Defaults	0.05	5.0
AGR "Agriculture"	0.05	5.0
FOR "Forestry"	0.02	5.0
OPP "Other primary sectors"	0.1	5.0
FOP "Food Processing"	0.1	5.0
MAN "Manufacturing"	0.1	5.0
ENE "Energy"	0.05	5.0
CNS "Construction"	0.1	5.0
TTR "Trade and Transport"	0.02	5.0
HOT "Hotel and restaurants"	0.1	5.0
EDU "Education"	0.3	0.0

User action:

- *Edit single fields with numerical values. Cut/Paste via clipboard possible*
- *Edit default values for all visible rows in the row "Defaults" and press "Apply defaults" button. The defaults for each columns will be copied over.*
- *Use the filter with * and ?? to select specific rows*

Example definition:

```
<control>
  <order>3110</order>
  <Type>table</Type>
  <Title>Trade elasticities</Title>
  <Value>0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,
    -0.5,-0.5,-0.5,-0.5,-0.5,-0.5,-0.5,-0.5,-0.5,-0.5,-0.5
  </Value>
  <Columns>Import supply,Export demand</Columns>
  <Rows>AGR "Agriculture"
    FOR "Forestry"
    OPP "Other primary sectors"
    FOP "Food Processing"
    MAN "Manufacturing"
    ENE "Energy"
    CNS "Construction"
    TTR "Trade and Transport"
    HOT "Hotel and restaurants"
    EDU "Education"
    OSE "Other services"</Rows>
  <range>0.1,5.0,0.1,-5.0,-0.1,0.1,</range>
  <gamsName>p_tradeElas</gamsName>
  <tasks>Calibrate CGE</tasks>
</control>
```

Notes:

- The range field might comprise several tuples of “low-up-increment” which will then be assigned to the columns of the tables. If there is only one tuple, it will be used for all columns.
- If a range is given, a spinner will be used as the cell editor and values outside the range will be rejected.

Output to GAMS:

```
PARAMETER p_tradeElas/
'test1'.AGR "Agriculture".'Import supply' 0.5
'test1'.FOR "Forestry".'Import supply' 0.5
'test1'.OPP "Other primary sectors".'Import supply' 0.5
'test1'.FOP "Food Processing".'Import supply' 0.5
'test1'.MAN "Manufacturing".'Import supply' 0.5
'test1'.ENE "Energy".'Import supply' 0.5
'test1'.CNS "Construction".'Import supply' 0.5
'test1'.TTR "Trade and Transport".'Import supply' 0.5
'test1'.HOT "Hotel and restaurants".'Import supply' 0.5
'test1'.EDU "Education".'Import supply' 0.5
'test1'.OSE "Other services".'Import supply' 0.5
'test2'.AGR "Agriculture".'Import supply' -0.5
'test2'.FOR "Forestry".'Import supply' -0.5
'test2'.OPP "Other primary sectors".'Import supply' -0.5
'test2'.FOP "Food Processing".'Import supply' -0.5
'test2'.MAN "Manufacturing".'Import supply' -0.5
'test2'.ENE "Energy".'Import supply' -0.5
'test2'.CNS "Construction".'Import supply' -0.5
'test2'.TTR "Trade and Transport".'Import supply' -0.5
'test2'.HOT "Hotel and restaurants".'Import supply' -0.5
'test2'.EDU "Education".'Import supply' -0.5
'test2'.OSE "Other services".'Import supply' -0.5
'test1'.AGR "Agriculture".'Export demand' 0.0
'test1'.FOR "Forestry".'Export demand' 0.0
'test1'.OPP "Other primary sectors".'Export demand' 0.0
'test1'.FOP "Food Processing".'Export demand' 0.0
'test1'.MAN "Manufacturing".'Export demand' 0.0
'test1'.ENE "Energy".'Export demand' 0.0
'test1'.CNS "Construction".'Export demand' 0.0
'test1'.TTR "Trade and Transport".'Export demand' 0.0
'test1'.HOT "Hotel and restaurants".'Export demand' 0.0
'test1'.EDU "Education".'Export demand' 0.0
'test1'.OSE "Other services".'Export demand' 0.0
'test2'.AGR "Agriculture".'Export demand' 0.0
'test2'.FOR "Forestry".'Export demand' 0.0
'test2'.OPP "Other primary sectors".'Export demand' 0.0
'test2'.FOP "Food Processing".'Export demand' 0.0
'test2'.MAN "Manufacturing".'Export demand' 0.0
'test2'.ENE "Energy".'Export demand' 0.0
'test2'.CNS "Construction".'Export demand' 0.0
'test2'.TTR "Trade and Transport".'Export demand' 0.0
'test2'.HOT "Hotel and restaurants".'Export demand' 0.0
'test2'.EDU "Education".'Export demand' 0.0
'test2'.OSE "Other services".'Export demand' 0.0
/;
```

Starting GAMS from GGIG

GGIG allows starting the GAMS project directly from the interface, either in compile or run mode. A break request can also be sent to GAMS (“stop GAMS”):

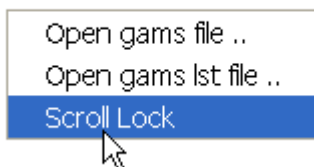


Once started, the GAMS project routes its output to the console back to lower right part of the interface:

```

--- .farm_constructor.gms (91) 3 Mb
--- exp_starter.gms (74) 3 Mb
--- .ini_herds.gms (19) 3 Mb
--- .title.gms (30) 3 Mb
--- .ini_herds.gms (86) 3 Mb
--- exp_starter.gms (78) 3 Mb
--- .decl.gms (29) 3 Mb
--- exp_starter.gms (195) 3 Mb
--- .title.gms (30) 3 Mb
--- exp_starter.gms (202) 3 Mb
--- .title.gms (30) 3 Mb
--- exp_starter.gms (240) 3 Mb
--- .store_res.gms (232) 3 Mb
--- exp_starter.gms (323) 3 Mb
--- .title.gms (30) 3 Mb
--- exp_starter.gms (325) 3 Mb
--- .store_res.gms (232) 3 Mb
--- exp_starter.gms (344) 3 Mb
*** Status: Normal completion
--- Job exp_starter.gms Stop 12/01/10 21:06:06 elapsed 0:00:00.047
GAMS RC 0
  
```

The pane with the content can be scrolled by a right mouse click in the pane to open a popup menu. If an editor is added under “opther options”, the GAMS and the listing file can be opened as well:

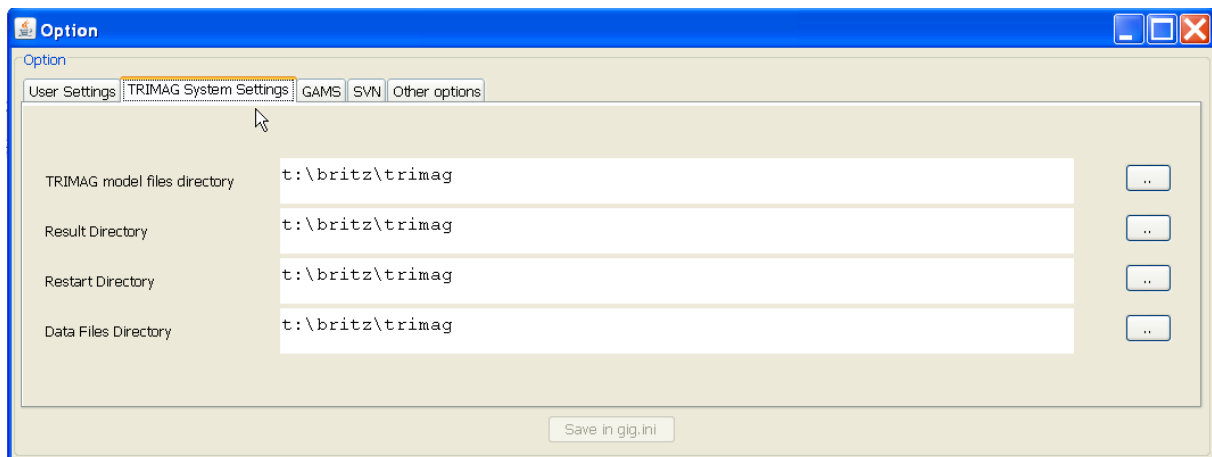


The pane can hence be “frozen” so that e.g. the status of a model solve can be inspected while the project continues to run. In order to successfully start a project, the ini file for GGIG must comprise the information where the GAMS executable can be found, but also where the GAMS code of the project to start is stored.

General interface settings

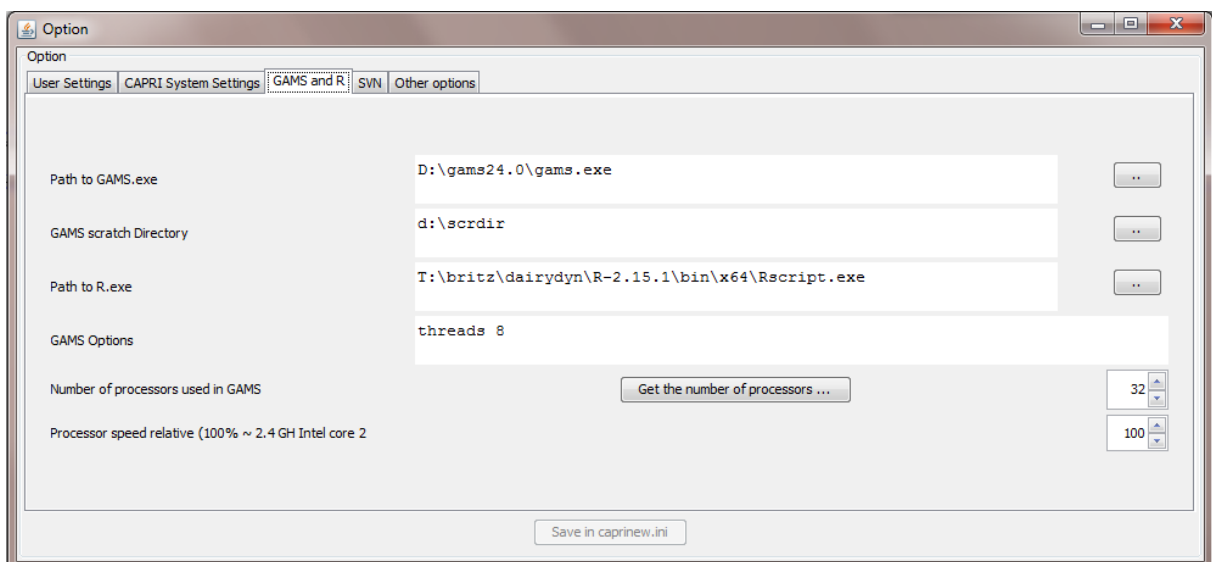
The interface has a few standard settings which can also be accessed via the “edit settings dialogue”. These are:

- Certain file locations:** the directory where GDX files for results are assumed to be stored (resDir), and three directories which can be used to adjust the specific model application: the root of the GAMS file (workDir in GAMS), called modelDir, a directory for restart files and one for data files.

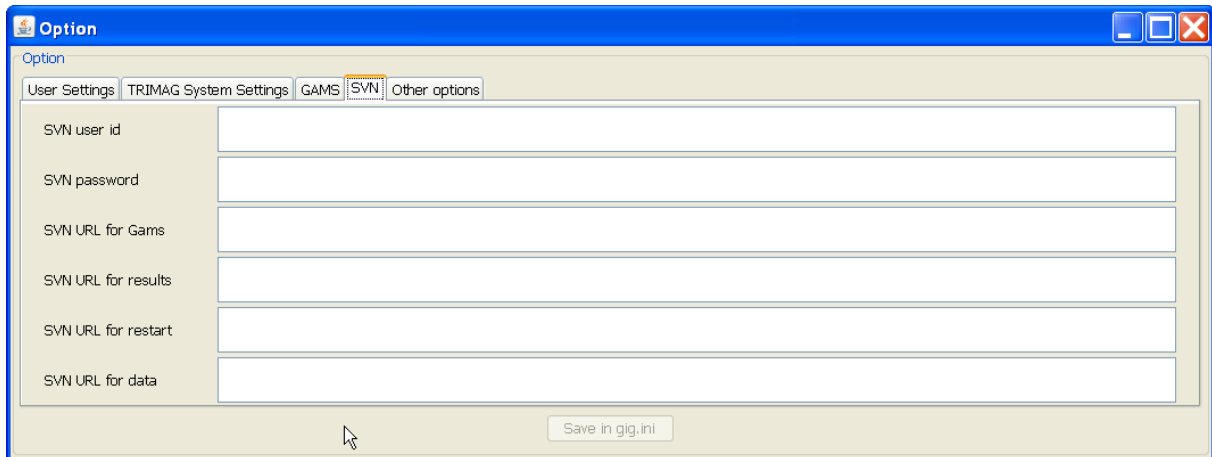


Note: The name of the system (here TRIMAG) is defined in the „GGIG.INI“ file

GAMS and R related settings

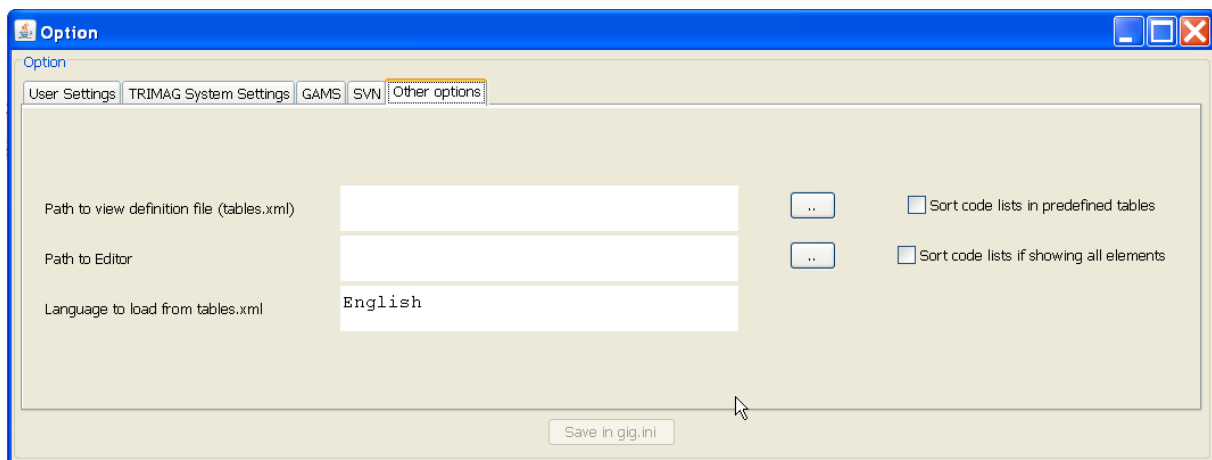


SVN related settings



The SVN settings can be used to perform checkout and updates in cases where the model code with related data, restart files or result files is under versioning control on a SVN server. If the model is not under version control, the settings “svn=no” renders the tabbed plan invisible.

Settings linked to the exploitation tools



Meta data handling

Why meta data?

Meta data are data about data. In many GAMS projects, it is impossible or cumbersome to tell exactly based on which shocks and settings results of a model run had been generated. That is due to the fact that run specific settings are not stored at all or not stored together stored with the results of the run. Later on, result users are often left guessing what exactly the settings underlying the run might have been.

In order to overcome that problem, the GGIG, drawing on [CAPRI GUI concepts](#), passes all interface settings, plus the user name and the current time, forward to GAMS in one SET called META.

A correctly defined interface with GGIG should allow to steer *all* run specific settings. If that is the case, the meta data generated by GGIG will provide an exact and sufficient definition of all run specific inputs, ensuring that all relevant meta data about a run are stored along with quantitative results in the same GDX file. Accordingly, GDX files shipped to other desks or committed e.g. to a SVN server still carry all necessary information to identify exactly the run.

Technical concept

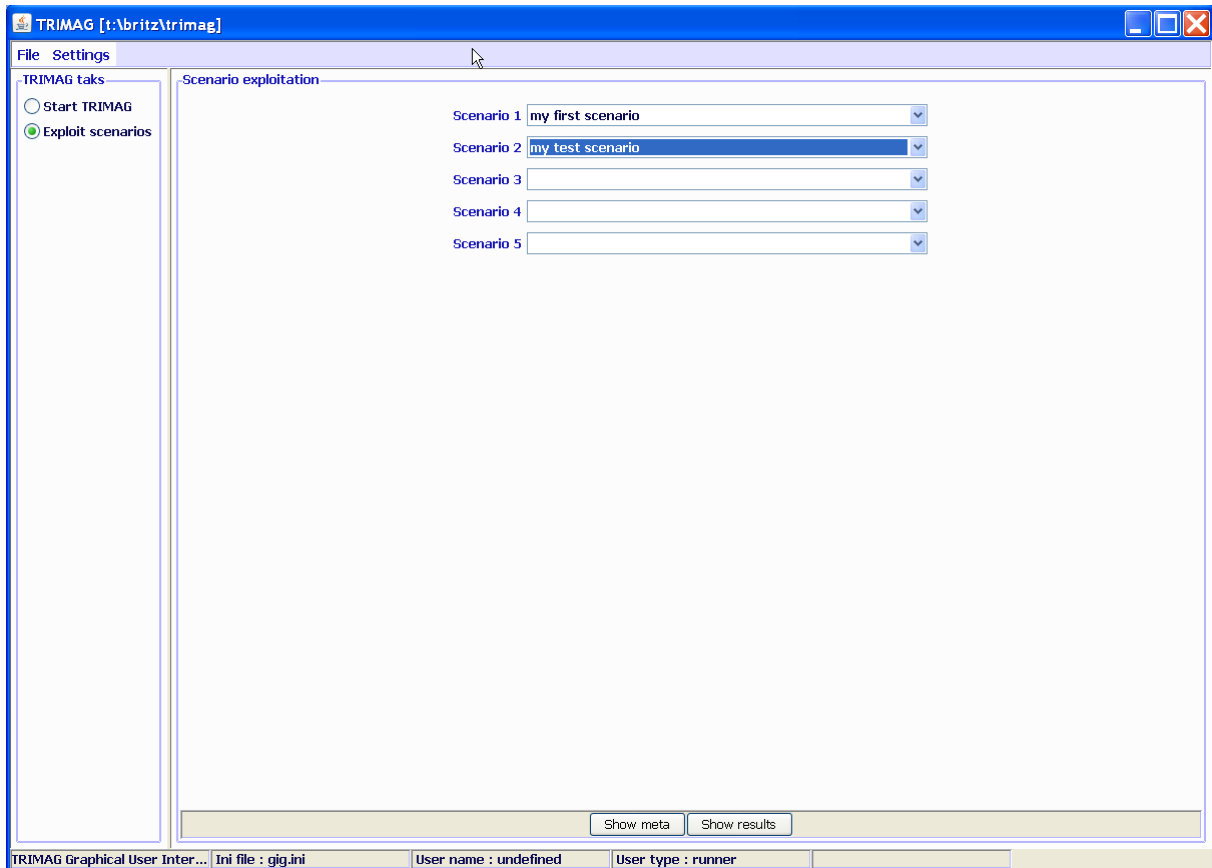
The meta handling is straight forward. The state of the different control is mapped into pairs of set elements and related long text descriptions as shown below from an example application:

```
SET META /  
'Scenario description' 'my test scenario'  
'Choose model type' 'CGE'  
'Relative weight flows' '30'  
'Use demand elasticities' 'true'  
'Set substitution elasticity' '6.0'  
'Countries' 'NL,'  
/;
```

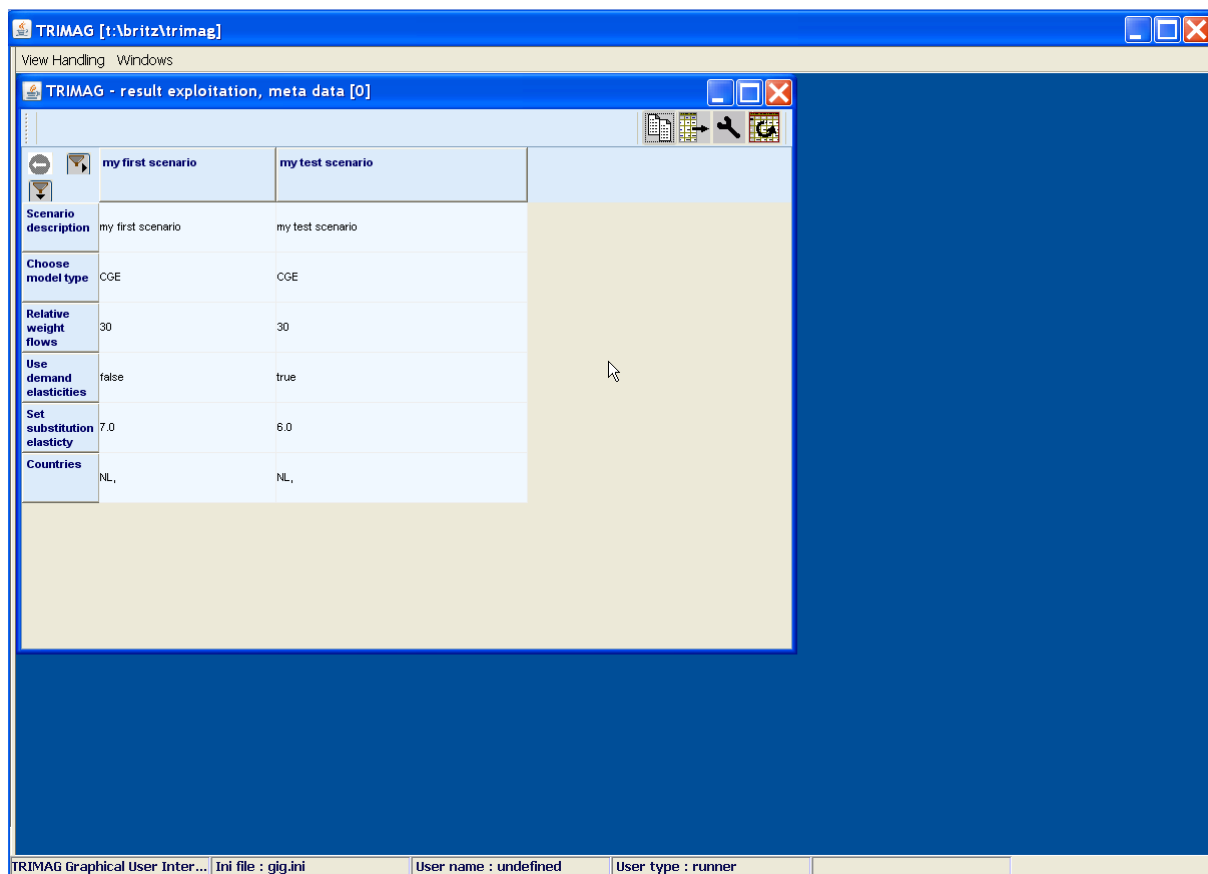
and, might with one GAMS statement as shown below, stored in the GDX files along with the results:

```
execute_unload "%scenario_description%.gdx" META,RESULT;
```

The user might then select some scenario:



And then, by pressing “show meta”, view the settings used for these scenarios:



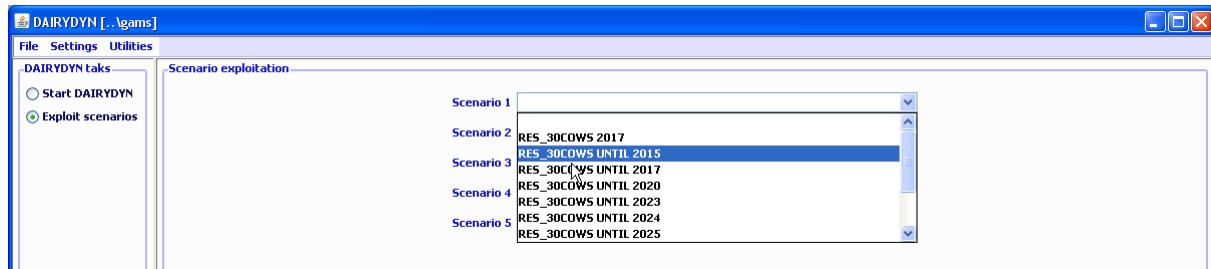
Exploitation

The basic strategy of the GGIG exploitation tools roots in the [CAPRI exploitation tools](#), which require that all model results are stored on an up to 10 dimensional cube, which is then stored in a GDX as a sparse matrix. Additional dimension can be added if several files are loaded, e.g. to compare scenarios or years. A specific XML dialect defines views (filters, pivots, view types) into the cube, and allows the user to load several result sets – typically from different scenarios – in parallel.

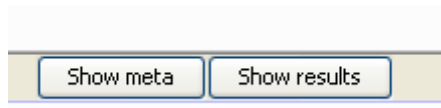
If no table definition file is present, GIGG offers a GDX viewer which some interesting possibilities not found in the standard GDX viewer (such as numerical sorting, statistics, selections). For details, the CAPRI GUI user manual should be consulted.

Selecting scenarios

When the user clicks on “Exploit scenarios” in the task selection panel, five drop boxes are shown on the right hand side. Each box comprises the list of GDX files found in the result director. The user can select in each box a file, or leave it empty.



At the bottom of the panel, pressing the “show results” button will open the exploitation tools



The screenshot shows the DAIRYDYN [..Agams] window with the 'Herd summary, mean [0]' table. The table has the following columns:

- Year: mean
- Farm: RES_30COWS 2017
- View type: Table
- Year: Year
- Farm: Farm
- View type: View type
- Table: Table
- actBased
- red0
- red1
- red2
- red3
- red4
- red5
- red6
- red7
- red8
- red9
- red10
- prodBased
- red0
- red1
- red2
- red3
- red4
- red5
- red6

		Herd size [Heads]	Revenues [Euro/ha]	Variable costs (incl. concentrates) [Euro/ha]	Costs of concentrates [Euro/ha]	Gross margin [Euro/ha]	Labour [hours/ha]	Milk yield [liter/head and year]	Nu lac [lit an]
actBased	red0	25.71	2980.48	451.46	201.29	2529.02	22.00	7.14	
	red1	25.68	2973.75	449.12	199.42	2524.63	22.00	7.13	
	red2	25.01	2971.81	448.56	199.08	2523.25	22.00	7.13	
	red3	24.24	2975.55	449.85	200.11	2525.70	22.00	7.14	
	red4	23.44	2980.59	451.59	201.50	2528.99	22.00	7.15	
	red5	22.67	2985.35	453.17	202.68	2532.18	22.00	7.16	
	red6	22.01	2986.51	453.38	202.64	2533.13	22.00	7.16	
	red7	21.35	2987.75	453.57	202.55	2534.17	22.00	7.15	
	red8	20.70	2989.09	453.73	202.35	2535.36	22.00	7.15	
	red9	20.04	2990.52	453.89	202.15	2536.62	22.00	7.15	
prodBas ed	red0	19.33	2992.08	454.18	202.11	2537.90	22.00	7.15	
	red1	25.71	2980.48	451.46	201.29	2529.02	22.00	7.14	
	red2	25.41	2940.22	437.50	190.10	2502.72	22.00	7.05	
	red3	24.75	2938.59	437.07	189.89	2501.52	22.00	7.05	
	red4	24.10	2937.43	436.64	189.51	2500.79	22.00	7.04	
	red5	23.43	2938.74	437.02	189.74	2501.71	22.00	7.05	
	red6	22.80	2937.10	436.33	189.04	2500.77	22.00	7.04	
	red6	22.13	2939.59	437.03	189.44	2502.56	22.00	7.04	

The full functionality is only available if a table definition file (see <http://www.capri-model.org/docs/Gui2010.pdf>, section on Editing the table definitions underlying the exploitation tools) matching the structure of the parameters in the GDX file is provided.

Menu bar

GGIG allows to add two types of menu items to the menu bar: HTML links and e-mail sent.

Design hints for structured programming in GAMS with GGIG

Using information passed from GGIG

As seen above, GGIG passes information mostly via \$SETGLOBAL settings. That has the advantage that the GAMS coder is rather free how to use the information. Take the following example (which could be generated from a slider):

```
$SETGLOBAL STEPS 99.0
```

There are several ways to use that information in GAMS code, below are a few examples:

1. Round the setting to an integer with \$eval in GAMS and use it in a set definition:

```
$eval steps round(%steps%)  
set step / S1*%STEPS% /;
```

2. Use it in an combined definition and declaration statement for a scalar

```
scalar s_steps / %STEPS% /;
```

3. Use it in assignment

```
|  
p_control("Steps") = %STEPS%;
```

4. Use it for pre-compiler conditions:

```
$eval steps round(%steps%)  
$ifthen %steps% == 1
```

5. Use for GAMS program controls

```
if ( %STEPS% > 10,  
);
```

Structure your program by tasks

The following example shows how the concepts of tasks can be used on conjunction with includes to structure a top-level program

```
* -----  
*  
* (4) ESTIMATE CONSTANT TERMS  
*  
* -----  
  
$iftheni "%task%" == "Estimate constant terms"  
$include 'est_const.gms'  
  
$endif  
*  
* -----  
*  
* (5) ESTIMATE CONSTANT TERMS AND TREND PARAMETERS  
*  
* -----  
  
$iftheni "%task%" == "Estimate constant terms and trend parameters"  
$include 'est_const_and_trend.gms'  
  
$endif
```

The basic idea is to have a common a part which is shared by many tasks and then blocks which perform task specific operations. As the “\$iftheni ... \$endif are working at compile time, not used code is excluded even from compilation which helps to save memory and reduce the size of the listing.

One entry points for run specific settings

A typical problem with more complex economic simulation models defined in GAMS is the steering of scenarios. GGIG pushes the GAMS developer to a code structure where all run specific settings are entered via the single include file generated by GGIG. That does not imply that all data for a specific scenario are comprised in the include file. It could e.g. mean that the user has selected via the interface the include file(s) with specific settings and that the names of these files are passed via the include file to GAMS.