GGIG Graphical Interface Generator

Programming Guide

Wolfgang Britz, August 2010

- Version March 2021 -

The following user guide documents 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 context of projects related to the [CAPRI](#) modelling system, which received 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 without charge.

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

Overview

The GAMS Graphical Interface Generator (GGIG) is a tool to generate a Graphical User Interface (GUI) for a GAMS or R project[1] with five main functionalities:

1. **Generation of user operable graphical controls from XML based definitions**. The XML file defines the project specific controls used and their layout. 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 ($SETGLOBALS, Set definitions, settings for parameters) or to R statements. It combines hence the basic functionality of a GUI generator and a rudimentary GAMS/R 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 or R project while passing the state of the control to GAMS respectively R** as an 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 GDX file into the GGIG exploitation tools. Note that GDX files can also be generated from R.

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 hence steered with xml-based 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 (cf. Britz and Kallrath 2012). 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,

---

[1] The code can also be used from inside Java, but that feature is not discussed in the documentation.

and do so typically quite efficiently. As a consequence, they seldom feel the need to develop 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, such as 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 (cf. Britz 2014a). Firstly, possible users are often not familiar with GAMS. But even with some elementary knowledge of the language, they might face problems understanding project specific GAMS code making use of advanced GAMS features. Secondly, they face the challenge to familiarize themselves with the specific code of the project (file structure, symbol names, set elements ...). In order to use the GAMS code for applications, a user needs to learn enough to know exactly which specific code changes are necessary to implement e.g. scenarios in a given project. Large and/or complex GAMS code of projects might basically exclude its 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 (Nilsson et al. 2008, Janssen et al. 2012). 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 (cf. Britz et al. 2015). 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 GUI design and then to code, debug and maintain 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. Such project specific GUIs are typically very powerful, but tend also to be too tool specific to be easily modified to fit to another (GAMS) project, and require an experienced software programmer for any changes in the GUI layout.

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 (Dol 2006). 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 direct competitor to GSE: GSE offers different 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, and user should first check carefully their requirements and what is offered by GSE or GGIG before taking a decision for one the two GUI generators.

An example of a completely different approach to a GUI for modelling tools offers SEAMLESS-IF (Rizzoli et al. 2009) 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. Furthermore, SEAMLESS-IF is 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, a Regionalized Agricultural Sector Model for Benin (Britz and M'Barek 2003), MIVAD, a hydro-economic model for a river basin in Morocco (Heidecke and Heckelei 2010) and the village CGEs developed in Advanced-Eval (Britz 2009). These GAMS based models respectively Java based GUIs used already before GGIG the CAPRI exploitation tools to inspect model results, 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 GGIG exploitation tool. It therefore pays typically off to start using GGIG for exploitation from the beginning (the same holds probably also for GSE and other GUI generators to be linked to GAMS or R code). 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, a GDX viewer or a batch processing mode.

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 let 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 coding GGIG.

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 provided by the project, 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 or R by an automatically generated include file which also contains automatically generated meta data. The user can also execute GAMS or R from the GUI. The GUI can equally load numerical results and meta data in a specific GDX viewer. The latter supports "view definitions", i.e. pre-defined reports to exploit the results. The details of the different elements are discussed below.
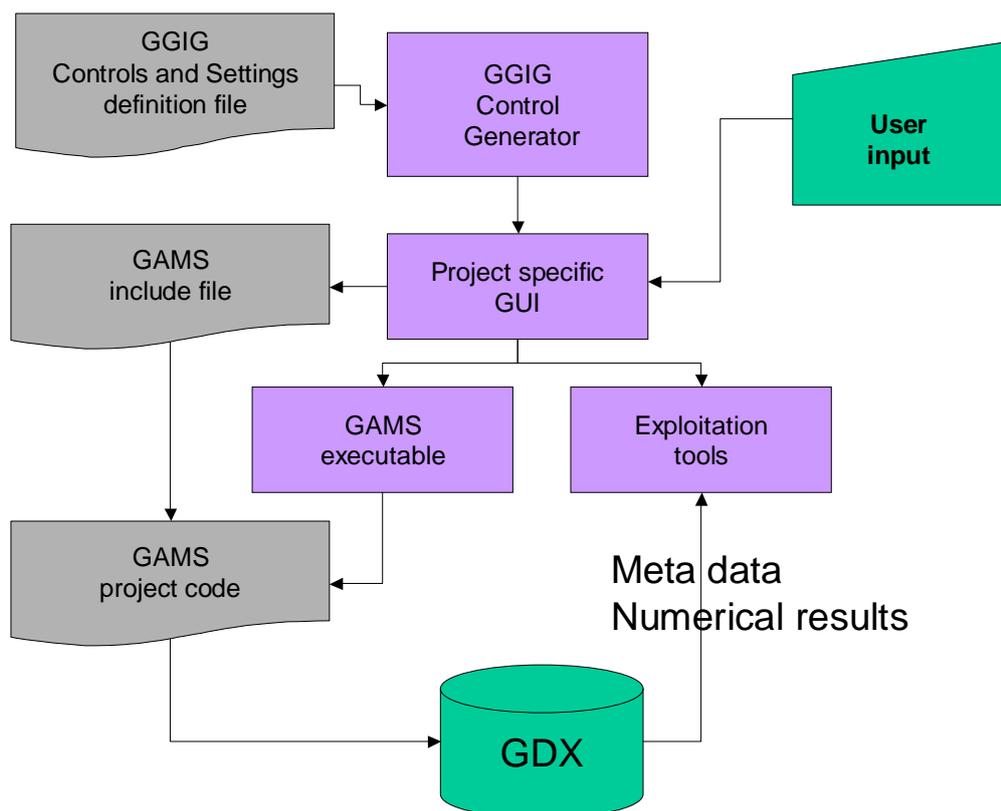
Diagram: Overview on information flow in GGIG

Current applications of GGIG

Since the first prototype, GGIG has been successfully implemented in a number of projects, examples are listed below in chronological order:

- FarmDyn: a fully dynamic single farm model (cf. Lengers and Britz 2012)

- 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, developed in the context of the NTM-Impact project (Wieck et al. 2012).

- A EU wide layer of regional CGEs with a focus on Rural Development measures on the second pillar of the CAP, which is now incorporated into CAPRI (cf. Schroeder et al. 2014).

- LANA-HEBAMO: A Hydro-Economic model for the lake Naivasha in Kenya (Kuhn et al. 2014).

- The FADNTOOL project which combines a set of economic tools for simulating policy impacts based on the FADN data.

- A modular and flexible Computable General Equilibrium model CGEBox

- An Agent Based model for structural change in agriculture, which is realized in GAMS, but uses GGIG as its user interface.

- BW-GLOBAL-FOR, a Multi-Commodity model for wood and wood based products (Britz et al. 2014)

- IFM-CAP by JRC-IPTS, PMP-based single farm models for the EU drawing on the FADN data base

- AGRISPACE, a recursive-dynamic agricultural sector model for Norway

- PEM by the OECD, a Multi-Commodity model linked to the OECD PSE/CSE data base

- METRO by the OECD, a CGE model with a focus on trade in value added

- The GLOBIOM model by IIASA

And last not least, the GUI of CAPRI (Britz 2014b) now is implemented in GGIG.

An overview on the GUI
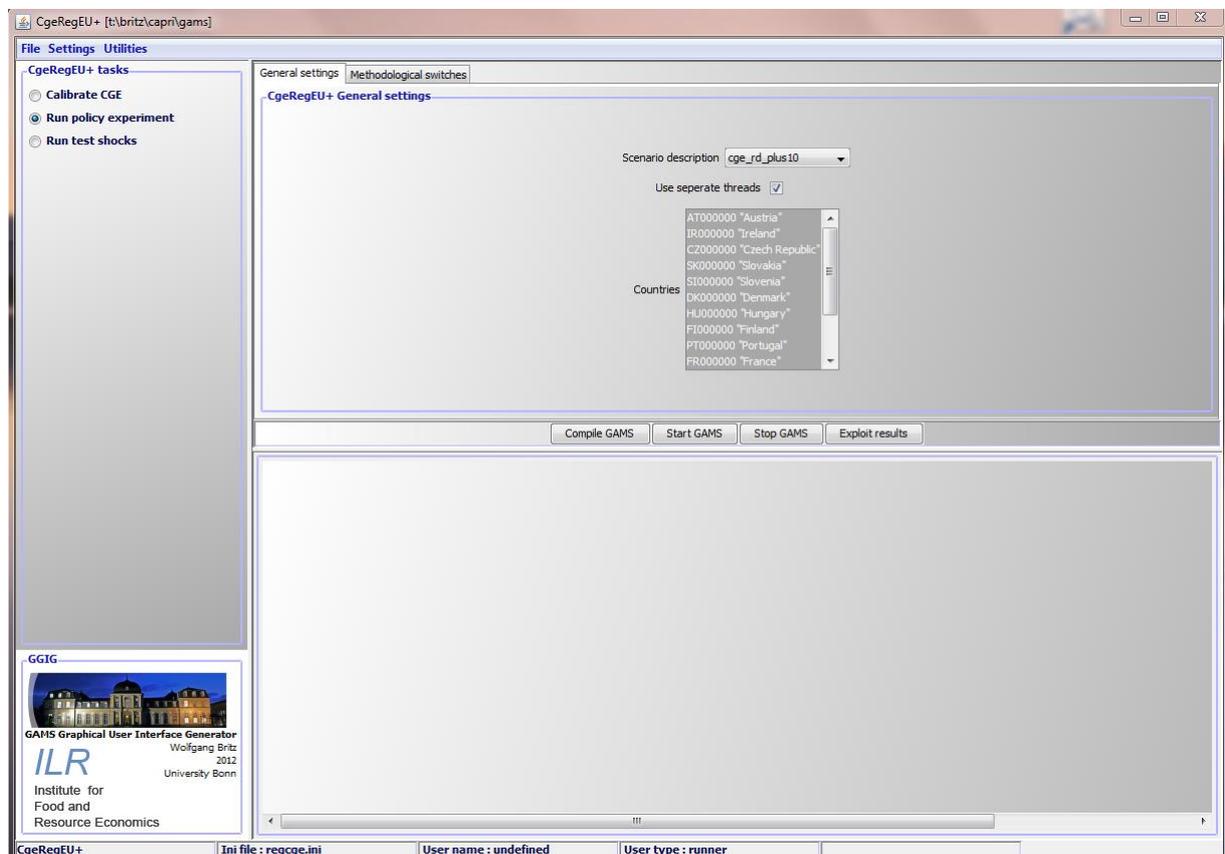
As shown in the example above, the GUI consists a few elements:

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 window in which the message log from GAMS is shown

    ii. **A panel to select data to view** and to start their 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 work steps 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 tasks, and each task might have its own GAMS or R process. That allows steering even rather complex tools which combine different GAMS or R projects based on one GGIG implementation. Splitting up a project into several tasks supports a structured development of the GAMS/R code as either separate GAMS/R files with a clear purpose are generated or a GAMS/R 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. Tasks can be combined into work steps to further structure the work flow in a project.

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, these graphical controls are used by the user to define textual and numerical settings which in turn define run specific settings for a GAMS/R 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 or R statements*, which can be included into a GAMS/R project to generate a specific run.

4. It allows *execution of GAMS or R*.

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 easy to use a text control in GGIG to enter directly the name of another file to include in the GAMS/R code.

The include file generated by GGIG which reports the state of the controls is overwritten each time the user starts the GAMS project.

Basic concept of the control definition file

GGIG supports two formats 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 defines the controls, tasks, etc. based on XML tags. As the XML file is parsed by a standard Java XML parser, 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 different tags (or keywords) are discussed in detail below.

Standard Java property files (deprecated)

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 files:

1. One default file with the control definitions and related default values. That file should be typically under version control.

2. A second file which is installation specific, it will solely store the values entered by the user on the interface and will be in the Java generic "ini" format. Its content is

updated each time the interface is closed, the next run will re-load the control and other setting from that file. Thus, that file should not be under version control.

A typical call will therefore look like:

Java Gig.jar project.ini project_default.xml

Where the first argument relates to the project "ini" file which stores the current user settings (it will be automatically updated when the interface is closed). The second argument defines the GGIG definition file. It is hence possible to host several GGIG based installations in one directory where the jars etc., are stored.

Tool name, logo, default directories and background color

The following three XML tags allow setting the tool name, the logo shown on the interface and the background colour:

```
<toolName><attr>FADNTOOL</attr></toolName>
<logo><attr>fadntool_logo.jpg</attr></logo>

<backgroundColor><attr>255,140,180</attr></backgroundColor>
```

Equally, the icon shown in the task bar can be set

```
<icon><attr>dairydyn.gif</attr></icon>
```

The option dialog comprises tabs for system directories and executables:

```
<defaultDirs><attr>model,res,dat,Policy editor</attr></defaultDirs>
<defaultExes><attr>GAMSExe</attr></defaultExes>
```


The default for the directories is "model, res, restart, dat". The abbreviation are kept for legacy reasons. As seen above, additional directories can be added. The XML definition file can refer to these settings in the definition of control, e.g.

```
<control>
        <type>fileselDir</type>
        <title>Scenario file</title>
        <value>base_scenarios\noshock.gms</value>
        <options>%modeldir%\scen\*.gms</options>
        <gamsName>shock</gamsName>
        <tasks>Simulation,Sensitivity analysis</tasks>
</control>
```

The "defaultsExe" field adds text fields and file selection button to the tab "GAMS" as shown below:
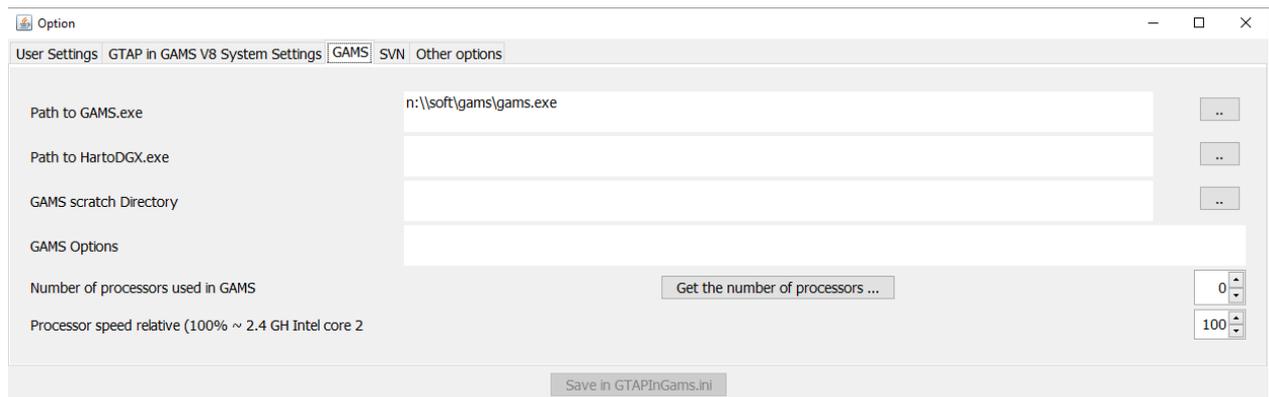
Worksteps

Worksteps allow to group tasks, and thus represent the top level of structuring actions in a tool. The following attributes are possible

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

Tasks   List of tasks

PDFLink   Context sensitive link as via a PDF bookmark

```
<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>
```

The work step selection is based on a set of radio buttons in a panel in the upper left corner of the generated GUI. It is not necessary to define work steps in a project.

Tasks

The control definition file must define a list of tasks (such as calibrating the model and running the model) for the project.

A task can have its own GAMS or R file to start, its own result directories and its own set of controls. Each control can however 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 in the order as they are defined in the control file on the left-hand side of the interface, below the work step panel (if work steps are defined):

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 (required)
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..dim9
filters	filters for scenario input, see below
xmltabledef	task specific view definition file in XML format
PDFLink	Context sensitive link as via a PDF bookmark

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

Tasks without a GAMS or R clearly cannot be executed, but they can be used to exploit GDX files. That allows to e.g. to explore different parts of data bases.

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 consists of 3 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. **"File_Starts_with"** will selected input files which match the selection of another control, such as in the following example:

    ```
    <dataSetSel><attr>Type=fileSel;Title=Dataset;Options=%datdir%\*.gdx;range=-1;tasks=none;</attr></dataSetSel>
    ----------------------------------------------------------------- 912 line(s) not displayed ----------------------
        <filter>Data set,dataSetSel,file_starts_with</filter>
    ```

    In the example above, a control with the same "Data set" is generated, comprising the file names found under "%datdir%*.gdx, and only results files

---

with a name starting with the name of the selected file will be shown in the drop-down boxes for scenarios:
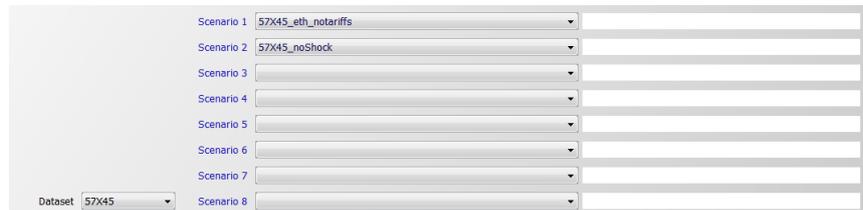
Note: the Java program will attempt to locate in the GUI folder a XML file with the name of the selected file and copy it to "generated.xml". That can be used to generate from GAMS a file with e.g. regional and product definitions matching a specific data base and use <xi:include href="generated.xml"/> in the view definition file to dynamically load these definition into the views.

c. 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 generated from 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 which records from the selected files are loaded in the viewer. In the example shown below, where the filter controls fit to the definitions above, only records where the region key starts with "BL" will be shown to the users.

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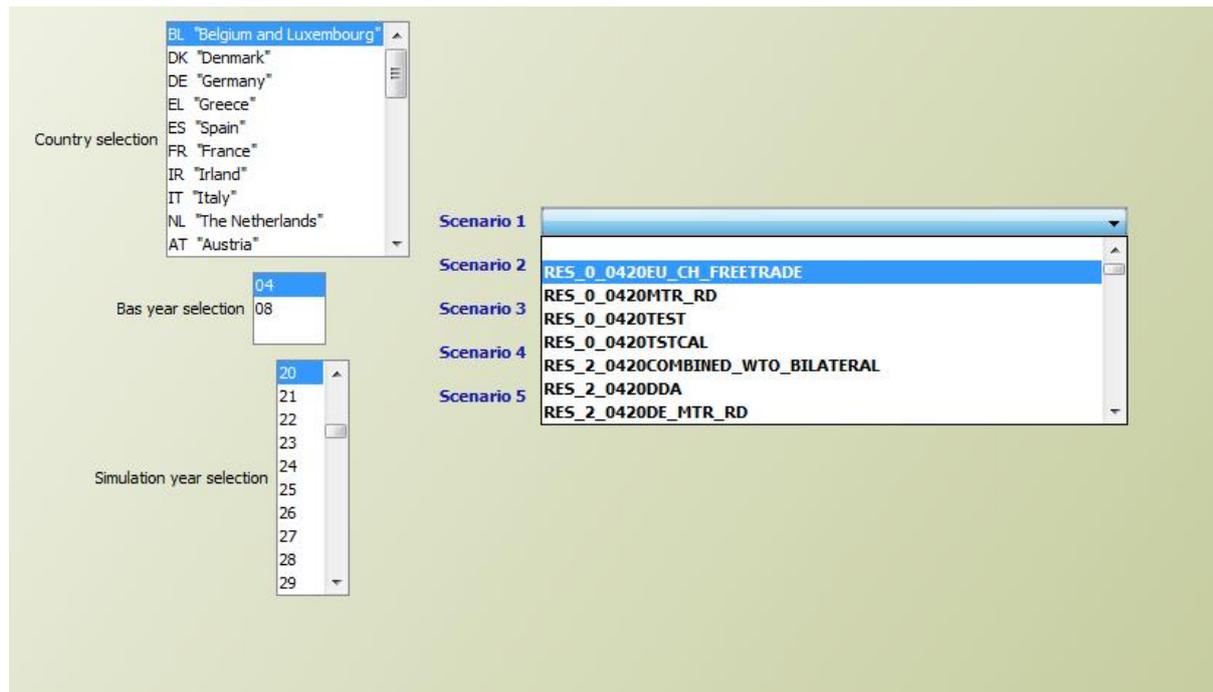
Normally, the name of the file will be used to characterize the "scenario". The "merge" is made for the case where several GDX files should be combined and the file name does not distinguish model runs. An example offers the downscaling component of CAPRI: it produces in separate GAMS run for the same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

Controls

Possible fields for controls

The necessary information for each control is stored in different tags for each control definition file. The controls are put on the interface in the order they are given in the XML. The following fields are available:

Type   defines the type of control (required). The different types are discussed below in detail.

Title   defines the description of the control as seen by user (required)

GamsName   defines the name of global settings resp. SET name (optional)

Value   pre-selected setting(s) (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 (optional)

Pdflink   Link to a pdf file and chapter to open on mouse over (optional)

Selgroups Selection list opened by pop-up menu (see Multilist control)

Style   Different style options (optional)

Disable   Control is blocked for input – useful to show settings on interface which are should be sent to GAMS for a specific task (optional)

dependsOn Defines inclusive O dependencies to other settings of other controls (optional)

dependsOnAll   Defines AND dependencies to other settings of other controls (optional)

Dependencies

The <dependsOn> and <dependsOnAll> fields allows to define dependencies between controls, i.e. under which settings of other controls the current control is visible or enabled.

The value of these fields comprises a list of trigger control names, and for each trigger the possible settings which render the current control visible respectively enabled. The controls are separated by a forward slash "/". The settings for each control are separated from the control name by a double point; the settings themselves are separated by comma. The settings describe an inclusive OR relation, i.e. the control is shown if one of the listed settings is currently chosen. For each trigger control, it can be defined if the dependent control should be rendered visible and enabled, and a default value can be set if the control is rendered invisible or disabled.

In the following example, the dependent control is triggered by the control named "Standard GTAP model". Only if the value of that trigger control is "false", the dependent control is enabled. Otherwise, its value is set to "comparative static".

```
<dependsOn>Standard GTAP model:false:enabled:Comparative static</dependsOn>
```

With the <dependsOn> field, the controls also are checked for inclusive or, i.e. if for one of the controls one setting matches, the current control is shown. The following example shows a tab which is only enabled if one of the listed farm branches is selected by the user:

```
<control>
        <Type>Tab</Type>
        <Title>Animals</Title>
        <tasks>Single farm run, Calculate MACs,Experiments dairy</tasks>
        <dependsOn>Farm branches:Dairy,Fattners,Sows</dependsOn>
</control>
```

The following example disabled a tab if a checkbox titled "GHGs" is not checked:

```
<control>
        <Type>tab</Type>
        <Title>GHGs and MACs</Title>
        <tasks>all</tasks>
        <dependsOn>GHGs:true</dependsOn>
</control>
```

The <dependsOnAll> field checks that for *all* controls at least one setting matches. In the following example, the panel is only shown if one of the listed farm branches *and* one of the listed crops are selected.

```
<control>
        <Type>spinner</Type>
        <Title>Grassland</Title>
        <Value>40</Value>
        <range>0,500,1</range>
        <options>int</options>
        <gamsName>nGrasLand</gamsName>
        <tasks>%allTasks%</tasks>
        <style>vAlignment:Spread,noOutputIfInvisible:true</style>
        <dependsOnAll>Determination of land endowment:Total arab and grass land/Farm branches:Dairy,MotherCows,Beef:any</dependsOnAll>
</control>
```

The <dependsOnAll> field can also be used to only show the control if several values from a multi-selection control are selected, by listing all necessary combination of the control name and the settings, e.g.

```
<control>
        <Type>panel</Type>
        <Title>Cereals</Title>
        <tasks>Single farm run, Calculate MACs</tasks>
        <dependsOnAll>Farm branches:Arable/Crops:WinterCere,SummerCere,MaizCorn/Farm branches:Dairy</dependsOnAll>
</control>
```

In the above case, the panel would only be shown if both the "Arable" and the "Dairy" farm branch would be selected, and one of the crops shows is chosen.

The values for the dependent control on visible tabs are always passed on to GAMS or R, even if it is currently not visible (or enabled). That eases writing the GAMS code as all $setglobals and sets defined via the interface for any combination of possible control settings are still declared. But clearly, the coder should only use these settings in the code if a default value is defined. Otherwise, settings from previous runs will be comprised in the include file which can neither been seen respectively changed by the user nor have a defined value.

In order to exclude that control values of invisible controls on active tabs are outputted to include, file use the style attribute "noOutputIfInvisible":

```
<style>vAlignment:Spread,noOutputIfInvisible:true</style>
```

In order to output also the values of controls on currently active tab, use the following attribute:

```
<inactiveSettingsToIncludeFile><attr>true</attr></inactiveSettingsToIncludeFile>
```

A specific type of dependency provides the population of tables, lists and combo boxes from GDX files. In case of such dependencies and a change in the state of the trigger component, the dependent component will be rebuilt

```
<dependsOn>Input file (*.gdx)</dependsOn>
```

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)

FileselDir   for 1 of n selections of a list of files found potentially in sub-directories, preceded by a sub-directory lists (two 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 or three-dimension parameter, comprises pivot possibilities (JTable)

SimpleTable   to enter floating point variables in a two or three-dimension parameter, no pivot possibilities (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:
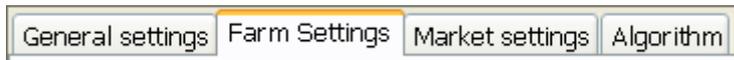
Remarks:

1. 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.

2. Tab names should be short.

Separator

Purpose

Used to structure the interface, gives a title for the next block of controls and keeps them together (top to bottom)

Applicable fields:

Title, Value, Tasks

Control optic:
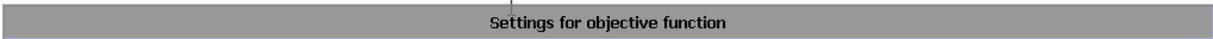
Example definition:

```
<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>
```

Panel

Purpose

Used to structure the interface, starts a block of controls which are ordered from top to bottom. Similar to separator, but not titled. One separator can span over several panels.

Applicable fields:

Tasks

Text

Purpose

Allows the user to enter text. Used e.g. to name the output generated by a run.

Applicable fields:

Title, Value, Tasks, Style

Control optic:

Scenario description | my first scenario

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
```

Note:

If the text entered refers to an existing file, it is recommended to use a filesel control instead.

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). Cannot be used for 1 of n selections where n > 2 – use a List instead.

Applicable fields:

Title, GamsName, Value, Tasks, Style

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
```

Note: True / False can be also outputted as ON / OFF to the include file for all controls. Use the following attribute:

```
<useOnOff><attr>true</attr></useOnOff>
```

Filesel / Dirsel

Purpose

Used for 1 of n selections of a list of files respectively directories. That is e.g. interesting when the user can choose from a list of pre-existing scenario definitions in GAMS files or to select a directory from which different data files are loaded.

Applicable fields:

Title, GamsName, Value, Options, Tasks, Style

Control optic:

Scenario description `cge_rd_plus10 ▾`

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 fill be automatically removed from the items.

User action:

tick / untick one of the selection possibilities with mouse

Example definition:

```
<control>
        <order>1010</order>
        <Type>Filesel</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
```

fileselDir / dirselDir

Purpose

Used for 1 of n selections of a list of files respectively sub-directories, potentially from sub-directories. That is e.g. interesting when the user can choose from a list of pre-existing scenario definitions in GAMS files, or to define sub-directories from which data will be loaded

Applicable fields:

Title, GamsName, Value, Options, Tasks, Style

Control optic:
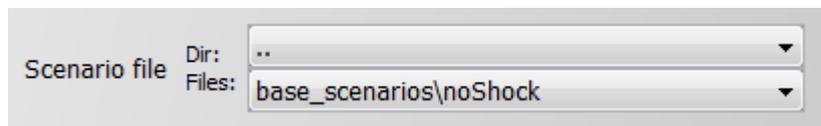
Note: Drop down lists will appear if the user clicks on arrow.

Possible value:

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

User action:

tick / untick one of the selection possibilities with mouse

Example definition:

```
<control>
        <Type>FileselDir</Type>
        <Title>Scenario description</Title>
        <Value>mtr_RD</Value>
        <Options>%modeldir%\pol_input\*.gms</Options>
        <range>0</range>
        <gamsName>result_type</gamsName>
        <tasks>Baseline calibration market model,
               Baseline calibration supply models,
               Baseline calibration farm types,
               HSMU baseline,
               Run scenario with market model, Run scenario only 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
```
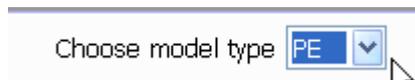
Singelist

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, Style

Control optic:

Choose model type PE ⌄

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`

Note:

- The user cannot deselect, i.e. one of the options is always active.

- The selection possibilities can also be loaded as a set from a GDX file,

```
<options>%datdir%\%%task%.Input file (*.gdx)%.gdx,setname=r</options>
```

. Note that a dependency should be added to ensure that options are properly updated if the user changes the file on the interface.

- Instead of using a GDX file defined via another control as shown above, the file name can also be entered as fix.

- Longtexts attached to the set-elements can be also loaded from the GDX file, use the style attribute "longtexts", i.e. <style>longTexts:true</style>.

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, Style

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

- A negative range will generate a number of rows, and define endogenously the number of columns such that all selection possibilities are visible, as seen below.

Possible values:

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>FAO2050 "Fao projections"
                GLOBIOM_EU "Projections with GLOBIO EU model"
                GLOBIOM_GL "Projections with global GLOBIOM model"
        </Options>
        <Value>FAO2050 "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).
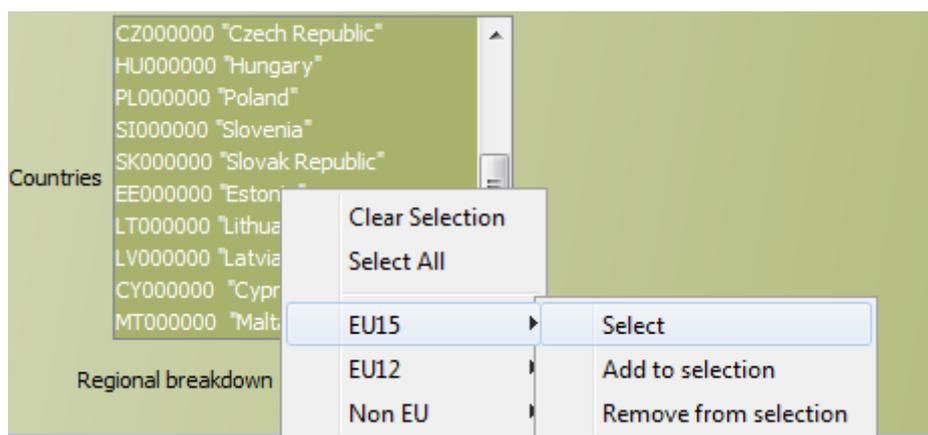
The control definition files can define selection groups which allow for groups of items to be selected, added or removed from the selection. Each selection group starts with a forward slash "/" following by the name of the group. The items are and the next selection group are

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 "Irland"
        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"
        LV000000 "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>
```

Note:

- The selection possibilities can also be loaded as a set from a GDX file,

```
<options>%datdir%\%%task%.Input file (*.gdx)%.gdx,setname=r</options>
```

- The standard output from the control is a set definition in GAMS, other output can be selected by:

```
<!-- The following style setting produces the _fst2 and _lst2 gams elements -->
<style>output:  fstlst set globals</style>
```

Radiobuttons

Purpose

Used to select for several items a one of n-settings, outputted as two-dimensional set

Applicable fields:

Title, GamsName, Value, Options, Rows, Tasks, Style

Control optic:

Possible values:

Defined by range field

User action:

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

Example definition:

```
<control>
        <Type>radioButtons</Type>
        <Title>Elasticity settings</Title>
        <options>Responding,Changing,Constant</options>
        <rows>
          K120QQ   "Wheat"
          K122QQ   "Rye"
          K123QQ   "Barley"
          K124QQ   "Oat"
          K126QQ   "Grain maize"
          K127QQ   "Rice"
          K129QQ   "Dry pulses"
          K130QQ   "Potatoes"
          K131QQ   "Sugar beet"
          OtherTP  "Other Crops"
          SE206    "Livestock"
          SE420    "Family Net Income"
          c        "Costs"
          l        "Land"
          cap      "Capital"
          lab      "Labour"
        </rows>

        <tasks>DEA</tasks>
        <columns>Responding,Changing</columns>
        <range>0,1,1</range>
        <gamsName>elasticitySettings</gamsName>

        <value>K120QQ  "Wheat".Responding,K122QQ  "Rye".Responding,K123QQ  "Barley".Responding,
        K124QQ  "Oat".Responding,K126QQ  "Grain maize".Responding,K127QQ  "Rice".Responding,
        K129QQ  "Dry pulses".Responding,K130QQ  "Potatoes".Responding,K131QQ  "Sugar beet".Responding,
        OtherTP "Other Crops".Responding,SE206  "Livestock".Responding,SE420  "Family Net Income".Responding,
        c       "Costs".Constant,l      "Land".Constant,cap     "Capital".Constant,lab     "Labour".Constant,</value>
</control>
```

Output to GAMS:

```
SET elasticitySettings(*,*) /
   K120QQ   .Responding
   K122QQ   .Constant
   K123QQ   .Constant
   K124QQ   .Constant
   K126QQ   .Constant
   K127QQ   .Constant
   K129QQ   .Constant
   K130QQ   .Constant
   K131QQ   .Constant
   OtherTP  .Constant
   SE206    .Constant
   SE420    .Constant
   c        .Changing
   l        .Constant
   cap      .Constant
   lab      .Changing
/;
```
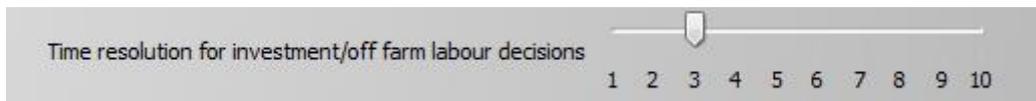
Slider

Purpose

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

Applicable fields:

Title, GamsName, Value, Options, range, Tasks, Style

Control optic:

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

Possible values:

Defined by range field

User action:

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

Example definition:

```
<control>
        <Type>slider</Type>
        <Title>Time resolution for investment/off farm labour decisions</Title>
        <Value>3.0</Value>
        <range>1,10,1</range>
        <gamsName>timeResolutionInv</gamsName>
        <tasks>all</tasks>
</control>
```

Output to GAMS:

```
$SETGLOBAL Set_substitution_elasticty 5
```

Spinner

Purpose

Used to select an value from a range of allowed ones. The increment can be chosen by the user.

If the range of the slider 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, Style

Control optic:

Use lower price iterations weights after iteration 20.0

Possible value:

Defined by range field (first, last, increment)

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 / TableSimple

Purpose

Define a table with floating point values passed to GAMS.

Applicable fields:

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

Control optic:

User action:

- *Edit single fields with numerical values. Cut/Paste via clipboard possible*

Example definition:

```
           <!--    wages        ,catt conc ,  oild, pigs conc  ,Diesel,ASS AHL  ,seed,  KAS,  PK,  KaliMag,Lime  Herb,Fung,Insect,gowth,water,hail -->
    <control>
          <Type>tableSimple</Type>
          <Title>Inputs</Title>
          <columns>Price,Growth rate "%"</columns>
          <value>
               1.0,  8.0,7.0,6.0, 32,28,0.001,200,220,240, 1150, 550,300,600, 0.70, 0.31,0.34, 1.00,  0.31, 0.29, 0.34,59.0, 1.00,1.00,1.00,  1.0,2.50,  9.34,
               1.0,  1.0,1.0,1.0, 1.0, 1.0,1.0,1.0, 1.0,1.0, 1.0, 1.0,1.0,1.0, 1.0,  1.0, 1.0, 1.0,  1.0,   1.0, 1.0, 1.0,  1.0, 1.0,1.0,   1.0,1.0,   1.0,
          </value>
          <rows>
               Wage rate full time "(Euro/hour)"
               Wage rate half time  "(Euro/hour)"
               Wage rate flexible hourly "(Euro/hour)"
               Maize silage "(Euro/t)"
               Gras silage "(Euro/t)"
               Manure "(Euro/t)"
               Cattle concentrate type 1 "(Euro/t)"
               Cattle concentrate type 2 "(Euro/t)"
               Cattle concentrate type 3 "(Euro/t)"
               Oils for feed
               Pigs concentrate type 1 "(Euro/t)"
               Pigs concentrate type 2 "(Euro/t)"
               Pigs concentrate type 3 "(Euro/t)"
               Diesel
               ASS
               AHL
               seed
               KAS
               PK_18_10
               KaliMag
               Lime
               Herb
               Fung
               Insect
               growthContr
               water
               hailIns
          </rows>
          <range>0.00000001,1500.,0,-5,+5,.1</range>
          <gamsName>p_inputPrices</gamsName>
          <tasks>Single farm run, Calculate MACs,Experiments dairy,Experiments arable</tasks>

    </control>
```

Notes:

- The value field gives first the default value for a column, next a set of default values for the rows for that column. Afterwards, additional columns follow the same scheme. It is therefore recommended to put all values for one column in a row to increase readability.

- 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.

- The content of the table can also be **loaded from a GDX file**, in which case the name of an GDX file must be entered under <options></options>, e.g.:

  `<options>grasAttr.gdx</options>`

- The rows, columns and dim3s of a table can also be loaded fom a GDX file, an example is shown below:

  `<rows>%resdir%/build/%%task%.Input file (*.gdx)%.gdx,setname=rNat</rows>`

- That case can be combined with dependencies by defining the GDX file from another field, see the example for single list.

Output to GAMS:

```
* Inputs
TABLE p_inputPrices(*,*)

*
                                    'Price'           'Growth rate '
 'Wage rate full time '            15.00
 'Wage rate half time   '          12.00
 'Wage rate flexible hourly '      10.00
 'Maize silage '                   30.00
 'Gras silage '                    24.00
 'Manure '
 'Cattle concentrate type 1 '     220.00
 'Cattle concentrate type 2 '     240.00
 'Cattle concentrate type 3 '     260.00
 'Oils for feed'                  1500.00
 'Pigs concentrate type 1 '       550.00
 'Pigs concentrate type 2 '       300.00
 'Pigs concentrate type 3 '       600.00
 'Diesel'                           0.70
 'ASS'                             0.31
 'AHL'                             0.34
 'seed'                            1.00
 'KAS'                             0.31
 'PK_18_10'                        0.29
 'KaliMag'                         0.34
 'Lime'                           59.00
 'Herb'                            1.00
 'Fung'                            1.00
 'Insect'                          1.00
 'growthContr'                     1.00
 'water'                           2.50
 'hailIns'                         9.34
;
```

Layout of the controls on the interface and style options for controls

Overall, the layout of the GUI is structured by the following main elements:

1. **Tabs** provide the uppermost layer, by showing different "pages" with controls.

2. **Separators** allow putting a heading over a group of controls. Controls under a separator share a common space and will be kept together as one block.

3. **Panels** allow defining a group of controls which are not separated if the interface is resized, i.e. a block of controls. A separator can span over several panels.

Generally, controls are put along the y-axis (top to bottom) into blocks. In order to provide an x-ordering (left to right), panels or separators can be used to define block of controls. If no panel and/or separator is given for a tab, all controls will be shown in one block.

Controls in a block stick together; the maximal height of a block on any page defines the minimum height of all pages. If the width of a page becomes too small, the rightmost block(s) of that page will move below of each all others, and the minimal height of all pages will increase. Furthermore, as discussed next, controls can share one line. Accordingly, it is recommended to build relatively small block of controls.

Note: if several panels are present under a separator, they are always laid out on along the x-axis. Reducing the width of the page will not reorder blocks below a separator in several lines.

Most of the controls allow for a style tag. Currently, only the following options are supported:

1. **Putting the controls in the same line** below the last one

```
<control>
        <Type>checkbox</Type>
        <Title>Allow for off farm work</Title>
        <Value>true</Value>
        <style>sameLine:true</style>
        <gamsName>allowForOffFarm</gamsName>
        <tasks>Single farm run, Calculate MACs, Experiments dairy,Experiments arable</tasks>
</control>
```

Which leads to the following:

Different states of nature for price ☐	Allow for off farm work ☑

The sameLine style tag can be combined with hidden or shortened control titles as discussed below.

2. **Left alignment**

```
<control>
        <Type>singlelist</Type>
        <Title>Base year</Title>
        <Value>2004</Value>
        <Options>2004,2006,2008</Options>
        <gamsName>BaseYear</gamsName>
        <style>vAlignment:left</style>
        <tasks>Define fts from FSS,
-------------------------------------------------------------------------------------
        </tasks>
</control>
```

1. **Left alignment of label and right alignment of control**

An example with two check boxes is given below:

```
<control>
        <Type>checkBox</Type>
        <Title>Allow purchases of silage maize</Title>
        <Value>false</Value>
        <gamsName>purchMaizSil</gamsName>
        <tasks>Single farm run, Calculate MACs,Experiments dairy</tasks>
        <style>vAlignment:Spread</style>
        <dependsOn>Farm branches:Dairy</dependsOn>
</control>

<control>
        <Type>checkBox</Type>
        <Title>Allow selling of silage maize</Title>
        <Value>false</Value>
        <gamsName>sellMaizSil</gamsName>
        <tasks>Single farm run, Calculate MACs,Experiments dairy, Experiments arable</tasks>
        <style>vAlignment:Spread</style>
        <dependsOn>Farm branches:Dairy</dependsOn>
</control>
```

| Allow purchases of silage maize | ☐ |
| Allow selling of silage maize | ☐ |
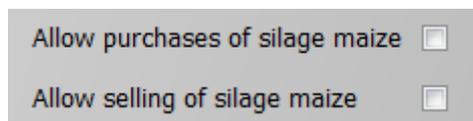
2. **Width and height** for tables and selection lists, e.g.

```
<control>
        <Type>TableSimple</Type>
        <Title>Soil</Title>
        <columns>Share</columns>
        <rows>l "Light",m "Middle",h "Heavy"</rows>
        <gamsName>p_soilShare</gamsName>
        <value>0.33,0.33,0.33,0.33</value>
        <range>0.0,1.0,0.01</range>
        <gamsName>p_soilShare</gamsName>
        <tasks>Single farm run, Calculate MACs,Experiments dairy,Experiments arable</tasks>
        <style>width:220</style>
</control>
```

3. **Positoning of the title on top of the control**: titlepos:onTop

```
<control>
        <Type>MultiListNotZero</Type>
        <Title>Farm branches</Title>
        <range>5</range>
        <Value>Dairy,Arable</Value>
        <options>Dairy,Sows,Fattners,Arable,Biogas</options>
        <gamsName>farmBranch</gamsName>
        <style>titlePos:onTop</style>
        <tasks>all</tasks>
</control>
```


4. Using line breaks in the title, removing the title or shortening the title:

It is sometime useful to use another title to be shown by the user as used internally to store settings. An example is the case where two neighbouring controls define a min and max value as in the following screen shot:

The XML-code is shown below. Note the "sameLine:true" for the second spinner which positions the two logically connected controls (min and max) in the same line. The second title is shortened to "max" based on the dedicated "title"-style. That allows to use "max" as a title for several of such cases whereas the full description is still used to store the settings internally (e.g. in a batch file).

```
<control>
        <Type>spinner</Type>
        <Title>Construction year of stables\n and machinery, min</Title>
        <Value>1995</Value>
        <range>1980,2010,1</range>
        <options>int</options>
        <gamsName>stableYearMin</gamsName>
        <tasks>Experiments dairy,Experiments arable,Experiments fattners,Experiments sows</tasks>
</control>

<control>
        <Type>spinner</Type>
        <Title>Construction year of stables\n and machinery, max</Title>
        <Value>1995</Value>
        <range>1980,2010,1</range>
        <options>int</options>
        <gamsName>stableYearMax</gamsName>
        <style>sameLine:true,title:max,valignment:left</style>
        <tasks>Experiments dairy,Experiments arable,Experiments fattners,Experiments sows</tasks>
</control>
```

If different style options are present, they should be separated by comma.

As a consequence of the layout concept discussed above, the following recommendations can be given:

1. Firstly, in order to allow for resizing of the overall the GUI, not too many controls should be placed on one tab.

2. Secondly, individual blocks of controls should be kept small to avoid that the page height becomes too large, they can be grouped using separators (<type>Separator</type>) and panels (<type>Panel</Type>.

3. Small controls, especially when logically related, can be placed on the same line with <style>sameline:true</style>.

4. Try to use short titles, if that cannot be avoided, consider introduced line breaks "\n" in the title description. Moving helpful comments into a <tooltip> can help to use shorter titles. Consider moving the title above the control using <style>titlePos:onTop</style>.

5. Thirdly, controls should be linked to tasks and user levels and dependencies between controls should be used where possible to hide not active controls.

Readability of controls can also be improved by using appropriate vertical alignment. The default case places the title left from the control and centres them jointly on the current panel. The alternative options (left alignment of title or left alignment of title and right alignment of control) where discussed above.

1. Treatment of long texts:

Lists can comprise long texts. This can either be outputted to the interface ("longTexts=true") or, as the default, are converted to tooltips as in the example below
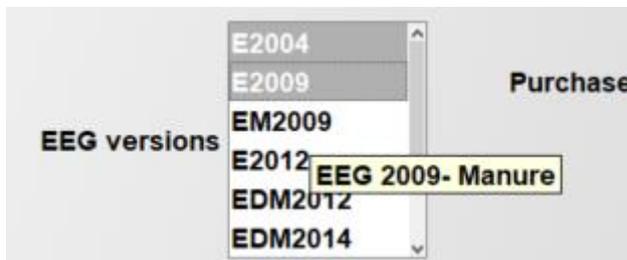
Help system

The help system consists of different elements:

---

1. Help related menu items (links to web pages, PDF documents) can be added to the menu bar.

2. Tooltips can be registered with controls

3. PDF Links, opened with the F1 help key, can be registered with tasks, worksteps and controls.

4. Similarly, PDF links and tooltips can be registered with views, see section on the exploitation part.

5. The exploitations part has fixed registered PDF links to the GUI user guide linked to various controls and dialogue.

Help menu items

GGIG allows adding three types of menu items to the menu bar: (1) HTML links, (2) e-mail sent items and (3) PDF links:

```
<helpmenuitem>
        <name>View capri web page</name>
        <value>http://www.capri-model.org</value>
</helpmenuitem>



<helpmenuitem>
        <name>Send mail to capri user list</name>
        <value>capritalks@ilr.uni-bonn.de</value>
        <type>mail</type>
</helpmenuitem>

<helpmenuitem>
        <name>GTAP in GAMS user guide</name>
        <value>..\doc\GTAPinGAMS with a GUI.pdf</value>
        <type>pdf</type>
</helpmenuitem>
```

Tooltips

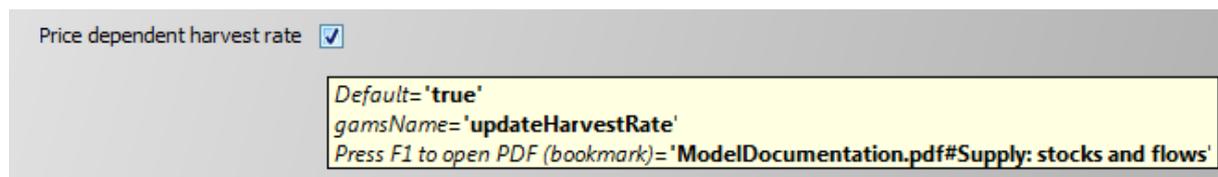GGIG shows as the default tooltip the default value of the control and its GAMS name:

Price dependent harvest rate ☑

```
Default='true'
gamsName='updateHarvestRate'
```

Further tooltip lines can be append with the <tooltip> field. HTML tags can be used to format these additional lines, e.g.
 to begin a new line, <B> to put text in bold or <I> to show text in italics. If HTML tags are used, the tooltip texts must be enclosed in with a <![CDATA[..]]> tag as shown below.

```
<tooltip><![CDATA[The new market layer defines an additional geographical layer of country aggregates<BR>
with homogenous border protection and market intervention measures (e.g. EU, MERCOSUR countries).<BR>
       If switched on, all trade policies (tariffs under trq, entry price system, intervention, flexible levies)<BR>
       are defined at the new geographical level.<BR>
       Be aware that before running a scenario, you need to re-calibrate the model with this option switched on.<BR>
       For more information contact Mihaly Himics.]]>
</tooltip>
```

Links to PDF documents with bookmarks

The <pdfLink> field, available for tasks, worksteps and controls, can be used to let the user open a bookmark in a pdf documentation when the help key F1 is pressed. The tooltip will inform the user if a PDF link is registered:

Price dependent harvest rate ☑

Default='**true**'
gamsName='**updateHarvestRate**'
Press F1 to open PDF (bookmark)='**ModelDocumentation.pdf#Supply: stocks and flows**'

If MS-Word docs are saved as PDF, so-called textmarkers can be generated (under options, unprintables are similar) from headings, the Adobe PDF generators offers the same functionality. The PDF files are defined relative to the GUI directory, i.e. in the example above, the files must be found in the GUI directory.

If the user presses the F1 key, the PDF documentation is shown in a separate window. The user can close the window; it opens with the next use of F1. The user can flip through the pages with page up/down buttons. The PDF decoder used internally is the non-commerical version which shows graphics not in full resolution.
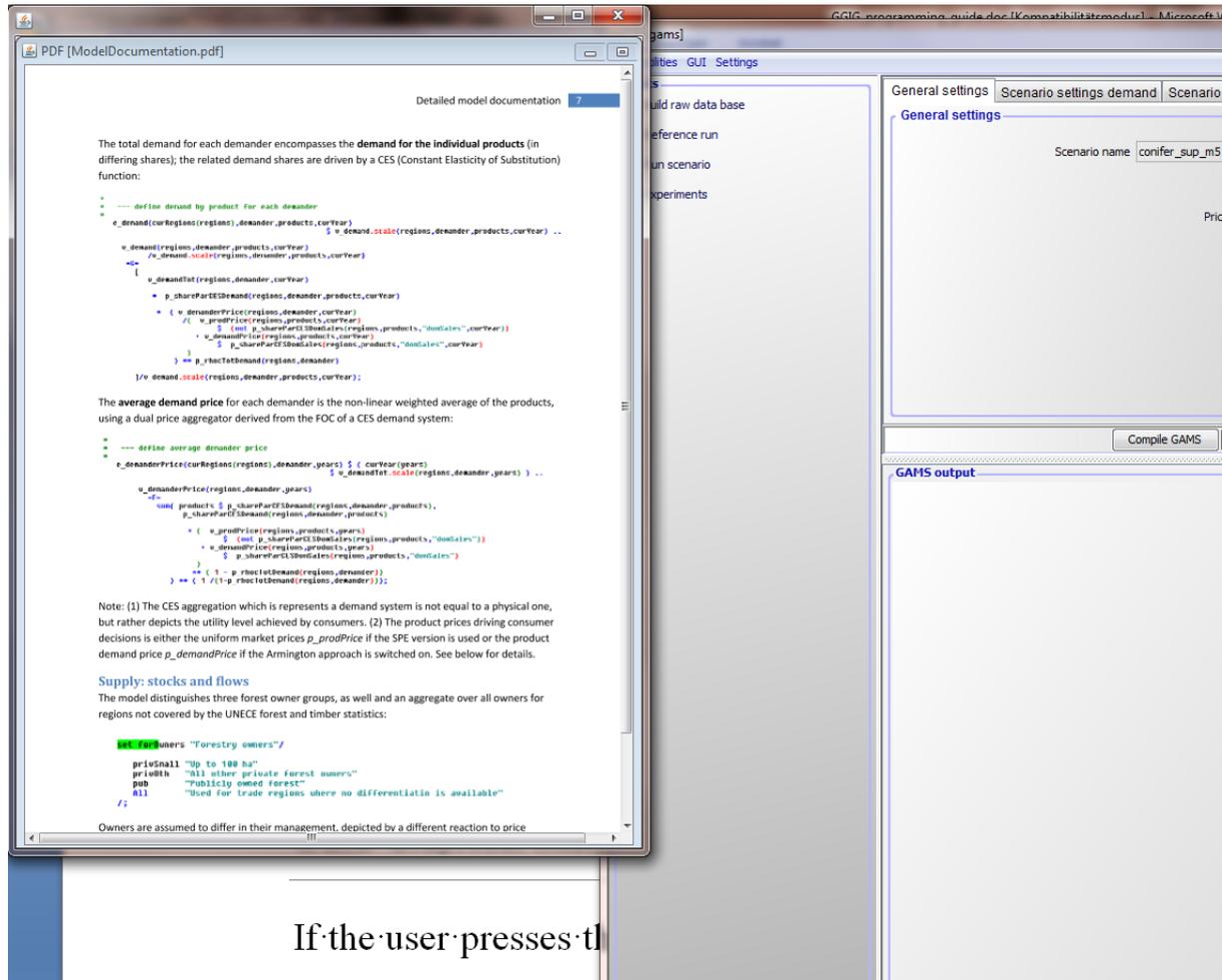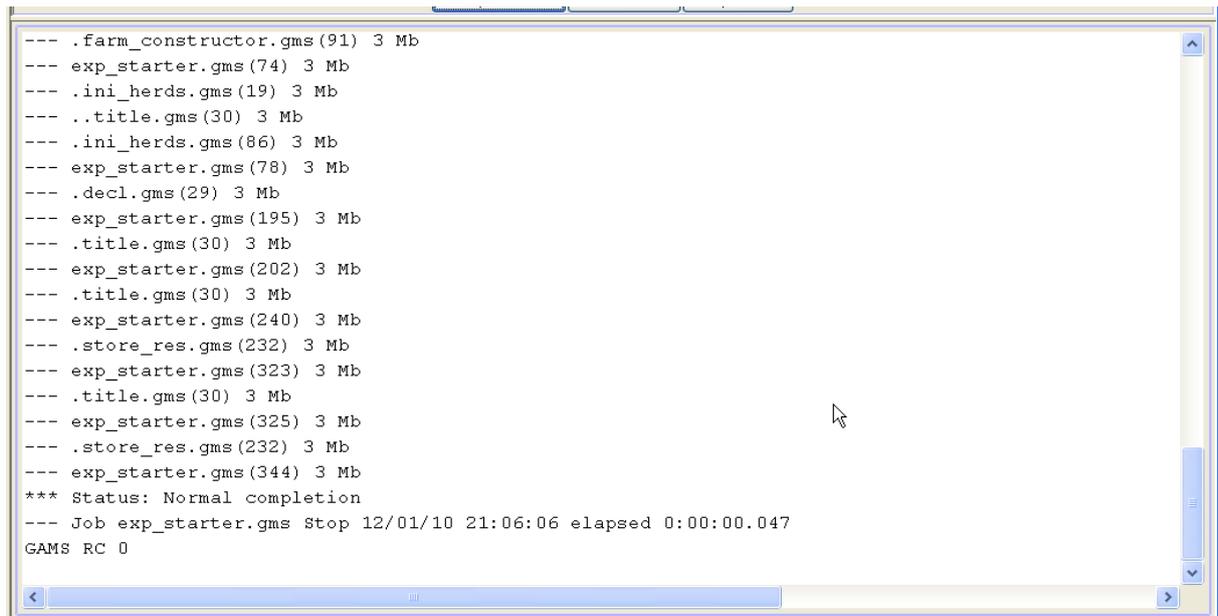
Starting GAMS from GGIG

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:

```
Open gams file ..
Open gams lst file ..
Scroll Lock
```
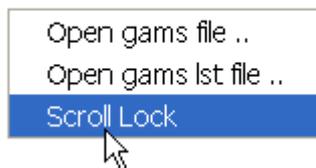
The pane can hence be "frozen" so that e.g. the status of a model solve can be inspected while the project continues to run. In order to successfully start a project, the ini file for GGIG must comprise the information where the GAMS executable can be found, but also where the GAMS code of the project 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.
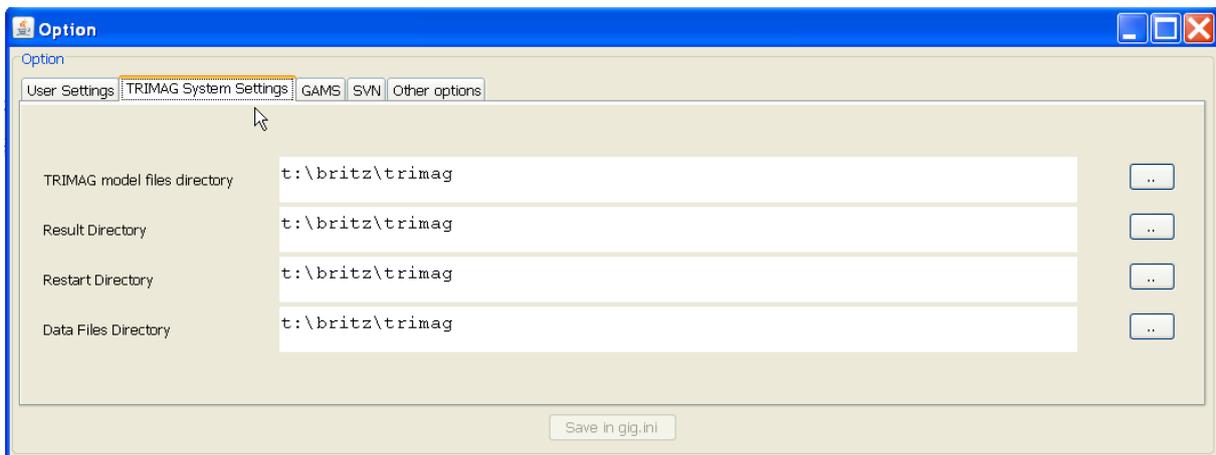
The default case shown above is set to "Model,Res,Restart,Dat", but the XML file can define only a subset of those or add additional ones, for instance
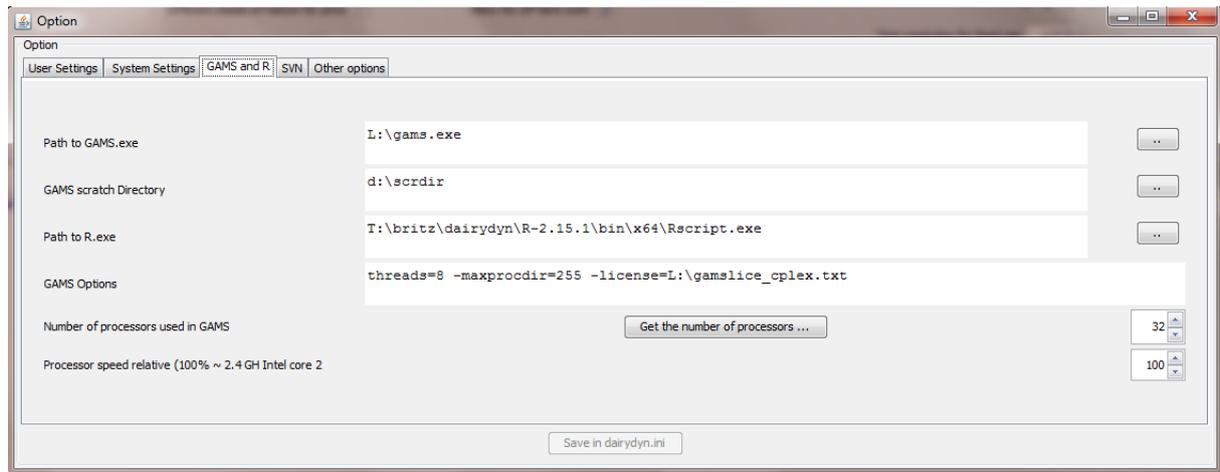
```
<defaultDirs><attr>model,res,dat,Policy editor</attr></defaultDirs>
```

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

Default settings can be defined in the XML file:

```
<datDir><attr>..\dat</attr></datDir>
<modelDir><attr>..\gams</attr></modelDir>
<resDir><attr>..\results</attr></resDir>
<restartDir><attr>..\restart</attr></restartDir>
```

GAMS and R related settings
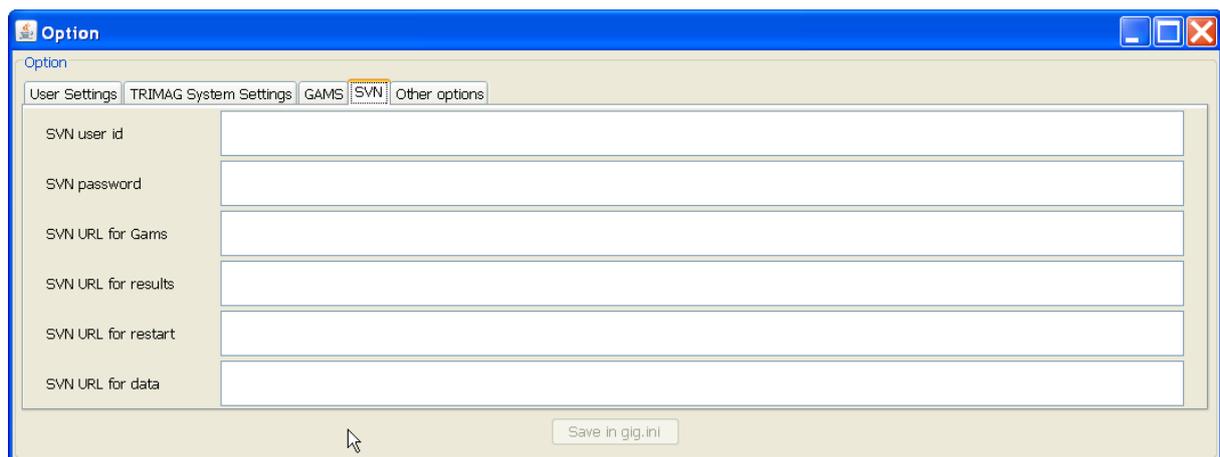

The default case is set to "GAMSexe,Rexe,Trollexe", but the XML file can define only a subset of those or add additional ones, for instance:

```
<defaultExes><attr>GAMSExe,HartoDGXExe</attr></defaultExes>
```

These settings can also be defined in the XML file:

```
<gamsexe><attr>D:\gams23.7\gams.exe</attr></gamsexe>
<rexe><attr>D:\r\bin\64\r.exe</attr></rexe>
<gamsoptions><attr>threads 0</attr></gamsoptions>
<numberofprocessors><attr>8</attr></numberofprocessors>
<processorSpeedRelative><attr>100</attr></processorSpeedRelative>
<scratchDir><attr>100</attr></scratchDir>
```

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.

Default settings can be defined in the XML file:

```
<SvnURLforDat><attr></attr></SvnURLforDat>
<SvnURLforGUI><attr>https://svn1.agp.uni-bonn.de/svn/capri/trunk/GUI</attr></SvnURLforGUI>
<SvnURLforGams><attr>https://svn1.agp.uni-bonn.de/svn/capri/trunk/gams</attr></SvnURLforGams>
<SvnURLforRestart><attr></attr></SvnURLforRestart>
<SvnURLforResult><attr>https://svn1.agp.uni-bonn.de/svn/capri/trunk/results/regcge</attr></SvnURLforResult>
```

Including credentials:

```
<SVNPwd><attr>1+\^.]</attr></SVNPwd>
<SVNUserID><attr>ierdlag_</attr></SVNUserID>
```
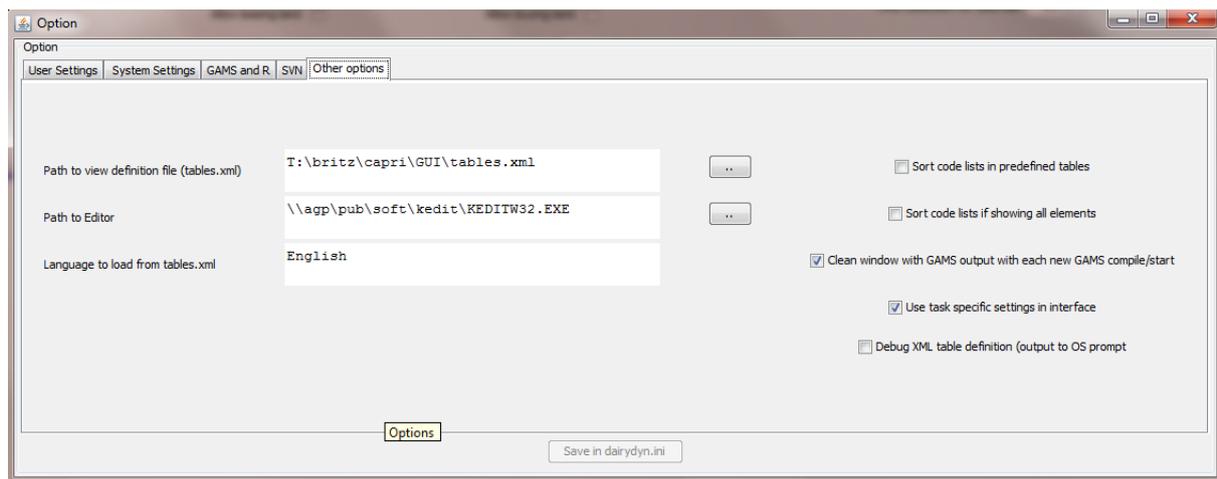
The credentials are obfuscated. In order to edit them, use the dialogue and copy the settings from the generated ini file to the XML.

It is also possible to switch the SVN panel completely off with:

```
<svn><attr>no</attr></svn>
```

Settings linked to the exploitation tools

The standard table file can be defined in the XML:

```
<xmlTables><attr>tables.xml</attr></xmlTables>
```

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 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](#), forwards all interface settings - plus the user name and the current time - 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 elasticty' '6.0'
'Countries' 'NL,'
 /;
```
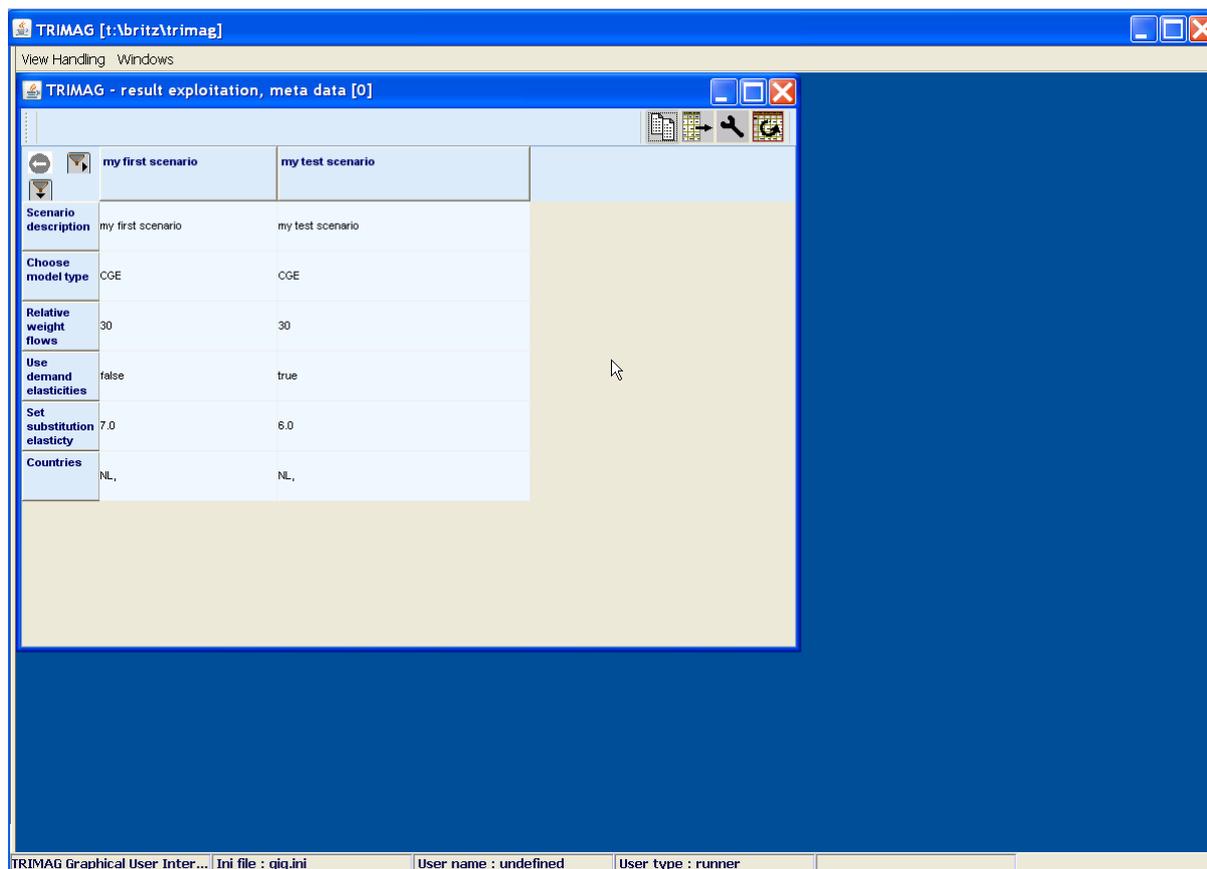
and, might with one GAMS statement as shown below, stored in the GDX files along with the results:

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

The user might then select some scenario:
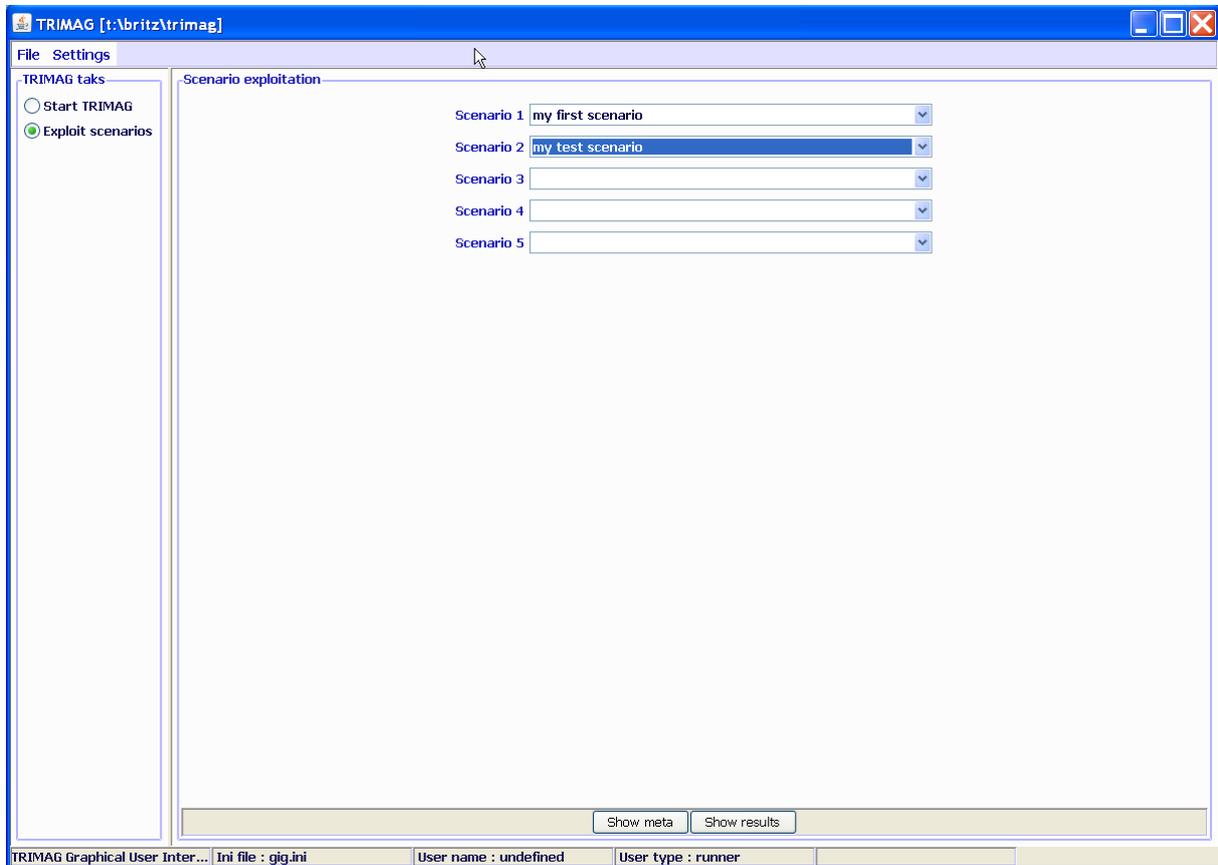
And then, by pressing "show meta", view the settings used for these scenarios:

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 container as a sparse matrix. Additional dimensions can be added if several files are loaded, e.g. to compare scenarios and/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 GGIG user manual should be consulted.

Excurse: history of GGIG, CAPRI GUIs

Reading the following chapter is not necessary to work with the GUI, but rather intended for a reader who is technically interested. The original software implementation of CAPRI was based on software available at ILR at that time and comprised a DBMS realized in FORTRAN with C/C++ code for the GUI. Whereas the economic model was from the

beginning implemented in GAMS, data transformations which generated the regional data base for CAPRI were still coded in FORTRAN, as it was done for the SPEL-EU data base serving as input for the regional data base of CAPRI. These FORTRAN routines were replaced by GAMS code in the CAP-STRAT project 2001-2004 along with new Bayesian based methodologies to ensure consistency and completeness.

The very first maps in CAPRI (in 1998) were produced with an MS-EXCEL mapping plug-in which was at that time a cost-free add-on. However, moving the data to EXCEL and then loading them in the viewer was not a real option for the daily debugging work on the data base and the model. Therefore, shortly before the first CAPRI project ended in 1999, a JAVA applet was programmed by W. Britz which was able to draw simple maps from CSV-Files. These CSV-file were automatically produced by the CAPMOD GAMS code. That code with slight modification remained active for quite a while, and some of the features are still found in the current mapping viewer. Then for a while, the CAPRI exploitation tools were based on XML/XSLT+SVG and a mapping viewer in SVG was realized. However, the XML solution had the big disadvantage of requiring a large amount of single ASCII input files, and was not really performant when complex pivoting was used. That XML-based solution was also used with other modeling systems, e.g. the @2030 system developed for the Global Perspective's Unit of the FAO.

To avoid that many ASCII files were generated, the next evolution step was a pure Java GUI. released around 2006, with direct access to GDX files which is the still the basis of the current Java code underlying GGIG. GDX files are an internal file format used by GAMS which allows a rather efficient I/O for large sparse tables. An API library delivered as part of GAMS installation allows to access GDX files from other applications. That design has the obvious advantage to be firstly based on the portable JAVA language. Secondly, as no external DBMS is used, it is possible to use CAPRI or other modeling systems applying GGIG by solely executing GAMS programs. Such a model might hence run on any system supported by GAMS, without the need to install additional software.

The next version of the CAPRI GUI consisted of three rather independent components. Firstly, a GUI to control the different work steps of CAPRI programmed in Java. That code dealt mostly with defining GUI controls (button, scroll-down lists etc.) to manipulate properties of CAPRI tasks, and then to start them as GAMS processes. That part has been thoroughly refactored with the revision of 2008. That refactoring introduced tasks as formal objects in the Java code of CAPRI, however in a far less generic way as it is done in GGIG. A

second important part is the CAPRI exploitation tool, which are basically generic enough to be used for other modeling systems as well. The 2008 refactoring left most of the code untouched compared to the code developed since 2006, with the exemption of the graphics which is now based on the JFreeChart library. However, as discussed below, in 2007, the mapping viewer was refactored in larger part to host the 1x1 km grid solution developed in the CAPRI-Dynaspat project. The exploitation tool is a rather unique solution to exploit result sets from economic models based on the definitions of views which are defined in XML tables. It combines features from DBMS reporting, data mining, spreadsheet functionalities and GIS into one package. And thirdly, there are some specialized pieces as the HTML based GAMS documentation generator which are linked into the GUI.

In 2010, a possible project by a Swiss team to add an interface to a GAMS project triggered the development of GGIG. A further motivation was the fact that any new GUI control needed on the CAPRI interface required changes in the Java code. That was not only tiring, it also let the Java code grow and made it harder to maintain. There were also increasingly features added to the CAPRI GAMS code which could not steered by the CAPRI GUI, but were switched on/off or otherwise modified by changes in the GAMS code. It was clear that this development was not sustainable. GGIG was first tested with very simple GAMS program, before the decision was taken to move the CAPRI GUI over to GGIG. That was probably a lucky development for GGIG, as CAPRI was at that time already a highly complex project with many tasks and settings, such that the further development of GGIG reflected the needs of complex model systems.

Execution of tasks via a GamsStarter and GamsThread

Execution of tasks in GGIG is handled by a *GamsStarter* object. An instance of *GamsStarter* lets the task write out the necessary include file(s) in GAMS format to generate a specific instance of the specific task (a simulation run for a specific scenario, simulation year, with the market model switched on or off …). A *GamsStarter* also knows about the working directory or other specific GAMS settings as the scratch directory. It may generate a pipe for the GAMS output to the console to show it in a GUI.

An *Task* can be executed by a *GamsStarter* who will then create a *GamsThread*. A *GamsThread* extends the *SwingWorker* interface of Java so that it may communicate with the normal event queue of JVM. A *GamsThread* can be gracefully terminated by sending a SIGNT signal to the GAMS process. That will let the GAMS execution stop at a specific

point determined by the GAMS engine itself and start the finalisation handling of GAMS as well as the removal of intermediate files and directories.

Refactoring the mapping part

When the 1x1 km grid layer was added to CAPRI during the CAPRI-Dynaspat project, it became obvious that the existing JAVA code used to produce maps in the CAPRI GUI needed some revision, especially regarding the way the geometry was stored. In this context, the question of using an existing GIS independently from CAPRI or the use of existing GIS classes plugged-into the CAPRI GUI was raised again and some tests with open-source products were undertaken. A stand-alone GIS as the sole option was certainly the less appealing solution. Firstly, it would have required exporting data from the GDX containers with CAPRI results to the GIS software, producing rather large intermediate files. It would also have left the user with the time-consuming and often error prone task of exporting and importing the data. Secondly, the user would need to switch between two different programs and GUI standards. And thirdly, all the usual problems with installing and maintaining additional software on a work station would occur. However, as indicated later, the GUI naturally allows passing data over to external applications and does hence not prevent the user from using a full-fledged GIS solution.

The main points taken into account during the search of a map viewing solution for CAPRI were: (1) possibility to import data from the CAPRI GUI efficiently, (2) user-friendliness, (3) performance and (4), in the case of plug-in libraries, expected realization and maintenance resource need, and naturally (5) license costs. It turned quickly that an ideal product was not available. Some of GIS products were not able to allow for the necessary link between newly imported tables with region codes and an existing geo-referenced geometry. Others had very complex user interfaces or produced run-time errors, took ages to draw the HSMU maps or were quite expensive. From the different options tested, the gvSIG (http://www.gvsig.com/index.php?idioma=en) freeware GIS seemed to be the only option, allowing the user to import data from a CSV – which must however be semi-colon delimited – and join one of the columns to a shapefile. At least the version installed at that time was however running not very stable.

In the end, it was decided to build on the existing code base and let Wolfgang Britz write the additional code "on demand". The main advantage of that approach is the fact that the mapping view is transparently integrated in the CAPRI GUI, it is sufficient to switch from

"Table" to "Map" in a drop-down list to produce a colored map, and that user demands regarding additional functionality may be and had been added, taking into account the specific needs of the CAPRI network.

Compared to ArcGIS, where the EU27 HSMU geometry plus codes and centroids requires about 340 Mbytes, the GGIG based version requires about 27Mbytes solely. Reading the GGIG coordinate information is somewhat slower compared to ArcGIS due to unzip on demand. The actual drawing operation takes about the same time as in ArcGIS (about 11 second for the full data set). Classification in GGIG is typically faster.

Views as the basic concept

The concept of the GGIG and former CAPRI exploitation tools is centred on the idea of a view. Content wise, each view may be understood as showing one or several indicators relating to results of GGIG tasks, e.g. in CAPRI, environmental effects of farming, prices or market balances. Technically, it could be understood as a combination of a selection query (i.e. filters in the different dimension of the data cube loaded by the exploitation tools) and presentation details (e.g. pivot, visualization as a map, table or graphic, fonts and colors), similar to a report definition in a DBMS.

Each view thus:

- extracts a certain collection of numerical values

- labels them so that they carry information to the user (long texts, units)

- chooses a matching presentation – as a table, map or graphic

- and arranges them in a suitable way on screen.

The views can be linked to each other, allowing a WEB like navigation through the data cube. Views can be grouped to themes. The user may open several views in parallel, and he may change the views interactively according to his needs, e.g. switch from a map to a tabular presentation, or change the pivot of the table, sort the rows etc.

Internally, each view is stored in an XML schema. The XML schema allows to attach long texts, units and tooltips to the items of a table, and thus to show meta-data information to the user. The XML schema hence replaces look up tables in a DBMS. It may equally store information regarding the pivoting, the view type (table, map, different graphic types), and for

maps, classification, color ramp and number of classes. The views can be grouped into logical entities, and are shown as a popup menu to the user.

Tabular views may feature column and row groups. Empty columns and rows can be hidden; tables can be sorted by column, with multiple sort columns supported. Numerical filter can be applied to columns.
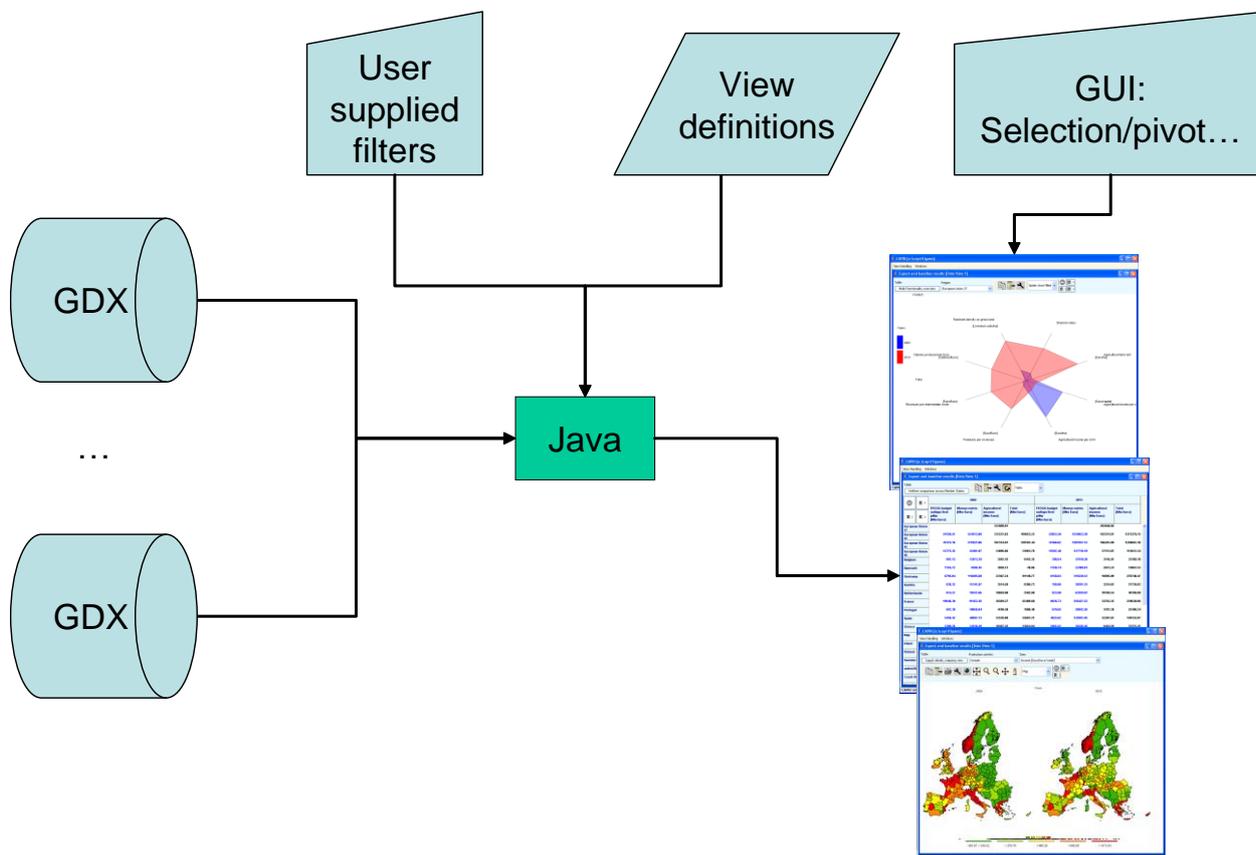
Data model

The underlying data model is very simple and straightforward. All data are kept in one large multi-dimensional sparse data cube, and all values must either be float values or strings. Each data dimension is linked to a vector of string keys. Those keys are the base for the filter definitions. Currently, data cubes with up to 10 dimensions are used (regions – activities – items – years – policy scenarios – dim5 ... dim9). The data storage model is equally optimised to the specific needs. As only float values or strings are supported, all non-zero data can be stored as one primitive array of either floats or strings. To allow fast and efficient indexing, a linear index is constructed from the multi-dimensional data cube, and the non-zero data and their indices are stored in a hash table. That renders data retrieval very fast. All data are loaded in memory at initialisation time: For moderately long linear indices about 10 Bytes are

required to store a non-zero float and its index as an integer. If the maximal linear index is very large, the index is stored as a long and the storage need goes up to about 16 Bytes. 20 Million non-zero numbers can hence be hosted in about 200 Mbytes.

The data are read from a generic file format generated by GAMS (General Algebraic Modelling System, a commonly used software package in economic modelling) called GDX, the software package on which CAPRI is based. Access to GDX is handled via an API provided by GAMS.

Client based solution

Technically, the exploitation tool is completely client based. That reflects the specific user profile of the CAPRI modelling system where the exploitation tool is integrated with an economic model and tools building up its data base. The main aim of the tool is to support forward looking policy analysis. For this purpose, users will create their own scenarios and in some cases even own variants of the export data, which will lead to processes requiring considerable processing and storage resources. A client-server solution where the production process and data storage would need to be hosted on a web server is therefore not a preferred solution, also as users will often develop variants of the modelling system by code modification in GAMS, and contribute to its development. The structure of the data driver would however very easily support linkage to a network or WEB based data bases. It should however be noted that the data base and GAMS code are managed via a Software versioning system, which is a kind of client-server environment. The reader is reminded that client-based does not exclude to store programs and data on file-server in a network environment.

The geometry model

The mapping viewer of GGIG is based on very simple and straightforward concepts. First of all, it basically supports solely polygon geometries, line strings (interpreted as rivers) and points for labelling. The storage model is optimised to host rectangles, and is especially efficient if the polygons vertexes are all points in a raster. The topology is not read from a shapefile, but stored in a generic rather simple format. However, a shapefile interface to generate the generic format is available. The vertices are stored in x,y coordinates, already projected in a rectangular coordinate system, and the viewer does not deal with the geographic coordinate system, but simply scales the rectangular coordinates in the viewport. The viewer in its current version solely supports one layer of quantities. Those restrictions naturally allow reducing memory needs, and, thanks to the rather simple data structures, also rather allow

performing drawing operations. It should also be noted that the JIT compiler of JAVA is indeed astonishingly fast given that that Java code is only precompiled.

The biggest topology currently handled simultaneously covers an intersection of Corinne Land Cover, slope classes and Soil Morphological Units and comprises around 2.7 Million polygons for EU27. As the majority of the polygons are rectangles, not more then 6-7 Million points needed to be stored. The topology handler and the drawing routines separate rectangles, for which only the two outer points are stored, from polygons, for which the vertices and centroids are stored.

The viewer is written in Java. Swing is used for the GUI in order to profit from the simplest implementation, the viewer has been written completely new, and is not based on existing GIS libraries. Even certain standard JAVA classes as e.g. for hash tables, have been replaced by own implementations, to reduce implementation overhead. Some care was given to support flexibility in classification, given that only quantities are supported, so that the tool covers natural breaks, quantiles, equal spread, mean standard and nested means. Area weighting is supported as well.

In order to export data to other applications, the tools support first of all tab delimited clipboard export, allowing import e.g. into EXCEL. Maps can be exported as JPEGs over the clipboard and to disk in several other formats. Alternatively, the user may export the data to an external file, in CSV format, DBF, to MS Access or to GAMS. DBF export will generate a second file comprising meta data.

The exploitation tools of CAPRI build on a rather simple structure. Each CAPRI work step stores its results as a GAMS parameter, representing a multi-dimensional sparse cube which is stored as a GDX file. The exploitation loads the non-zeros from one or several GDX files into memory. However, given the length of the different dimensions and the use of short codes, the user would be typically lost on his own in the large tables. The XML definition file is the equivalent of a collection of "SQL queries" as it defines views which combine filters in the dimensions of the cube with information on how to show the results (pivot, table, graph or map).

XML definitions for views

Why a XML definition files for views?

The exploitation tools of GGIG build on a rather simple structure. Each GIGG task can store its results as GAMS parameter representing a multi-dimensional sparse cube and save it on disk as a GDX file. The exploitation loads the non-zeros from one or several GDX files into memory. However, given the length of the different dimensions and the use of short codes, the user would be typically lost on his own in the large tables, which can comprise several million non-zero data and basically an unlimited amount of zero cells. The XML definition file defines the views explained above, and allows a structured and user-friendly way to exploit the results of the different work steps. It also separates raw data from the views and from the GUI code itself, which requires relatively little information about the underlying data and their structure besides what is provided by the definition files. XML is an industry standard to store structured information in non-binary text files, which explains why that format was chosen.

As explained in more detail in the excurse above, each view can be understood as to define a report, i.e. a combination of a selection query and information on the presentation of the selected values.

The structure of the XML definition files for the views

General comments

The XML parser used by the GUI's java is a standard XML parser. The table viewer currently supports up to 10 dimensions, which are named as:

1. Region
2. Activity
3. Product
4. Scenario
5. Year
6. Dim5, Dim6, Dim7, Dim8, Dim9

in the XML-file. These "logical dimensions" which are used in the XML definition file can be mapped to any dimension of the original data cube read in by the java code. Pivoting can then

be used to map these "logical" dimensions to viewport dimensions seen by the user such as the columns and rows of a table.

Necessary tags for tables

A table definition is found between the <table> … </table> tags. It must at least define:

- The table theme, such as <theme>Welfare</theme>. The themes are shown as a drop-down menu in the exploitation tools.

- The table name, such as <name>Welfare comparison between Member States</name>. The names must be unique.

- The items of the tables and the dimension where they stem from (or that non item dimension is used).

The order of the themes and table names will define their order in the drop-down menu. Alternatively, the <themes> tag can be used to order the themes e.g.

```
<themes>Meta,Scenario shifter,
Welfare,CAP,Policy indicators,
Prices,Markets,BioFuels,Trade,Farm,Farm - totals,Farm EU,
Feed,Fertilizer,Environment,
CGE Meta,CGE Policy,CGE Accounts,CGE Price,CGE Production,
HSMU,DNDC,Multi-Functionality,Energy</themes>
```

Defining the items of the table

The underlying idea of having a "hand defined" list of items for one of the definitions stems from the observation that most tables have only a very limited number of columns, and that these are normally formatted with care regarding their text comprised. A table therefore typically comprises a definition of items, but the items must not necessarily be mapped in the column viewport.

<item>

<itemName>Money metric</itemName>

<key>CSSP</key>

<unit>Mio Euro</unit>

<longtext>Consumer welfare measurement: expenditures necessary to reach utility in current simulation under prices of reference scenario</longtext>

<link>Money metric</link>

</item>

An item definition is enclosed in the <item>…</item> tags. It must at least comprise a <key> and an <itemName> tag. The case sensitive key must match the symbol identifier as found in the GDX file, whereas the itemName can be freely chosen.

Facultative tags are:

<unit>: a physical unit shown in table

<longtext>: a text shown when the mouse hovers of the column

<link>: a link to another table for the table cells under the column.

<colormode>: the color mode used when a map is drawn for the item. The following modes are supported:

- GYR   Green Yellow Red

- RYG   Red Yellow Green

- GR   Green Red

- RG   Red Green

- BG   Blue Green

- GB   Green Blue

- WB   White Black

- BW   Black White

- LD   Light Blue Dark Blue

- DL   Dark Blue Light Blue

<eval>: the item is calculated from other items, e.g. <eval>VAL + VAL[*,BlueBox,*,*,*,*] + VAL[*,DeMinimis,*,*,*,*]</eval>

<node>: the item defines a vertex in a network graph

<edge>: the item defines additional an edge in a network graph

In order to define the dimension from which the items are taken, the user can set either:

<itemDim>region</itemDim>

Deprecated is the old style:

<isActivity>NO</isActivity>

It is also possible to refrain from declaring any items:

<itemDim>none</itemDim>

In which case only the filters used on the different dimension are active. The possibility to attached attributes to each item is lost in that case.

Which means that the table loops over the products, and the items refer to the activity dimension. A typically example is a table with market balance elements: items such as "FEDM" are found in the columns of the CAPRI tables where also the activities are stored. Consequently, the table will loop over the products, and not over the activities. Alternatively:

<isActivity>YES</isActivity>

allows only items from the product dimension, and lets the table loop over the activities. A typical example provides a table showing activity levels, yield or economic indicators for the production activities.

Tables can also be defined solely on filters without any specific item dimension:

```
<table>
  <theme>Trade</theme>
  <name>Exports</name>

  <regionSel>all</regionSel>
  <productSel>all</productSel>
  <activitySel>trade|transCost|tariff_regex</activitySel>
  <hideEmptyRows>yes</hideEmptyRows>
  <hideEmptyCols>yes</hideEmptyCols>
  <dim5Sel>reg</dim5Sel>
  <dim5Text>Importing region</dim5Text>
  <itemDim>none</itemDim>
  <defPivot>0RSD</defPivot>
</table>
```

Additional tags: <subTheme>

Allows introducing sub-themes in the table selection.

Additional tags: <defpivot>

The definition of the default pivot for a view consists of a setting as shown below

```
<defpivot>rowGroups=product;rows=dim7;columnGroups=;columns=scenario;</defpivot>
```

Deprecated old style

The deprecated old style to define the pivot string consists of 4 characters. The first character position is for the table row blocks, the second for the table rows, the third for the column

blocks and the last for the columns. The logical dimensions are labelled with the following characters:

A   Activity

D   Dim5

I   Items

M   Activity and Product merged

P   products

Q   Year and dim5 merged

S   Scenario

R   regions

X   Region and dim5 merged

5...9   Dim5 ... Dim9

0   Empty

The definition <defpivot>0R0S</defpivot> thus means: regions are in the rows, scenarios in the columns. The definition <defpivot>PISR</defpivot> puts the products in the row blocks, the items in the rows, the scenarios in the column blocks and the regions in the columns.

Additional tags: <defview>

Defines the default view used for the tables, the list of default views is equal to what the user can select in the drop-down box:
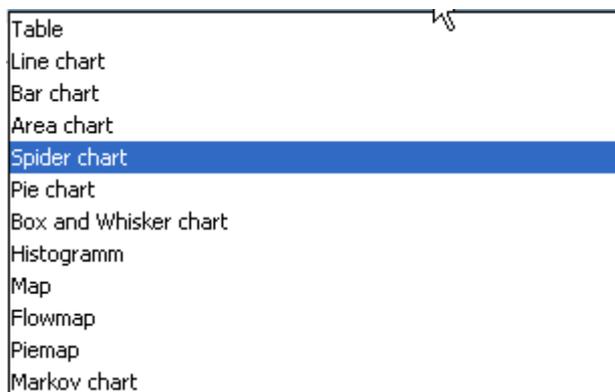
Additional tags: **<COO>**

This tag defines the geometry to use for maps. Currently, the following geometry files are available in CAPRI:

NUTSII.zip   NUTS 2 geometry for countries covered by the supply module

MS.zip   NUTS 0 geometry for the countries covered by the supply module

RMS.zip   Global geometry for the regions with behavioural functions in the market model

RM.zip   Global geometry for the trade blocks in the market model

HSMU.zip   1x1 km pixel clusters for EU 27 without Malta and Cyprus

There are also 1x1 km pixel clusters for individual Member States, but these are internally passed to the viewer when only one country is shown.

Alternative texts for the dimensions

Normally, the names for the dimensions are passed to the view by Java. However, their name can be changed by:

<regionText>….</regionText>

<activityText>…<activityText>

<regionText>…<regionText>

<productText>…<productText>

<scenText>…<scenText>

<dim5Text>…<dim5Text> ... <dim9Text>…<dim9Text>

<yearText>…<yearText>

That text is shown:

- As description above the outer drop-down selection boxes:

Region		Years	
European Union 27	⌄	2013	⌄

- In the pivot dialogue:

- And in gaphics / map titles and the like.

Additional tags: <clone>

The tag uses the item and other definitions from another table, and can be used e.g. to show the same selection in a different pivot or view types, e.g.:

```
<table>
  <theme>Welfare</theme>
  <name>Welfare comparison, overview across regions</name>
  <clone>Welfare overview</clone>
  <defpivot>0R0S</defpivot>
</table>
```

The clone tag *must* immediately follow the name tag, as otherwise, preceding definitions are lost.

Additional tags: <drop>

That tag is uses in conjunction with a <clone> and allows removing an item from the cloned table definition:

```
<table>
    <name>Income, per capita</name>
    <clone>Income</clone>
    <drop>Private consumption price index</drop>
--------------------------------------------------------------------------------
</table>
```

Further tags

There is a longer list of further tags which refer e.g. to definitions of graphs. They are here only listed in here with their default settings without a detailed explanation:

```
"FontSizeRelative","60",
"zerosAsMissingValues","false",
"blueishColors","false",
"showMarkers","false",
"domainGridLinesVisible","true",
"rangeGridLinesVisible","true",
"showAxisTitles","true",
"ShowlastColumn","true",
"commonRange","false",
"autoRangeIncludesZero","true",
"nBins","0",
"quantile","50",

"spiderChartMaxAxis","6",
"spiderChartMaxObs","5",
"SpiderChartFilled","true",
"SpiderChartForegroundTransparency","10",

"pieChartMaxPlots","10",
"pieChartMaxPies","25",
"pieChartLabelMinimum","5",
"PieChart3D","true",
"PieChartSimpleLabels","false",
"PieChartCircular","false",
"PieChartForegroundTransparency","20",

/* The maximum number of bar plot with their own value range axis */
"barChartMaxPlots","4",
/* The maximum number of bar groups (= elements on the domain axis, taken from the table rows. */
"barChartMaxDomains","10",
/* The maximum number of bars in a bar group, taken from the table column groups. */
"barChartmaxSeries","5",
"barChart3D","true",
"barChartStacked","false",
"barChartVertical","true",
"BarChartCylinder","false",
"BarChartForegroundTransparency","10",
"barChartDrawOutline","false",
"barChartDrawShadow","false",
"barChartFilledBars","true",

"lineChartMaxObs","25",
"lineChartMaxPlots","5",
"LineChartMaxSeries","10",
"LineChart3D","false",
"LineChartVertical","true",
"strokeWidth","2f",
"LineChartDrawLines","true",
"LineChartDrawShapes","true",
"LineChartForegroundTransparency","0",
"lineChartCumulative","false",
"lineChartSort","false"
```

The following list further tags, partially explained above:

```
"theme","null",
"subtheme","null",
"name","null",
"clone","null",
"item","null",
"isactivity","null",
"itemdim","null",
"regionsel","","activitysel","","productsel","","yearsel","","scensel","",
"dim5sel","","dim6sel","","dim7sel","","dim8sel","","dim9sel","",
"regiontext","","activitytext","","producttext","","yeartext","","scentext","",
"dim5text","","dim6text","","dim7text","","dim8text","","dim9text","",

// to handle later as props of AgpDataView
"defpivot","",
"defview","",
"fractiondigits","-1000",
"codesandkeys","null",
"comparedim","null",
"compareitem","null",
"comparemode","null",
"hideEmptyRows","null",
"hideEmptyCols","null",
"COO","null",
"pdf","null",
"evalall","null",
"language","null"
```

Filters for the elements in the different dimensions

Without filters, all elements found on a logical dimension will be shown to the user in any table. The exemptions are the items defined for a specific dimension, see above. In order to restrict the selection in the other logical dimensions, a selection list can be defined in the table definition. Take as an example the following XML tag:

<regionSel>MS,RM<regionSel>

It means that the table will only show elements with the tag <region> (see below) which comprise MS or RM in their <sel> field. The example would refer to the Member States. There is a specific selection list:

<regionSel>FromDataCube<regionSel>

Which will neglect the elements under <region> as defined in the file, but rather takes any one found in the data cube. The option was introduced originally for CAPRI to avoid the necessity to define all 180.000 HSMU codes in the file.

Alternatively, a regex string can be used, e.g.

<dim5Sel>red[0-9]+_regex</dim5Sel>

In both cases, the code will nevertheless try to locate matching entries in the predefined lists based on their keys.

Attaching long texts and filters to elements

Items for activities, products, regions and dim5 are typically defined in the file, see the following example:

  <region>

    <key>SK020038</key>

    <itemName>SK020 - FT41 / GT100 - Specialist dairying (FT 41)</itemName>

    <sel>[all, RS, SK, FA, SKFA, FT41, GT100, FT41GT100]</sel>

  </region>

The definitions for one item are enclosed in the tag (<region>…</region>, <activity>…</activity>, <product>….</product>, <dim5>…</dim5>).

The order of the items in the tables is defined by these lists.

Each item has a key, which corresponds to the symbol identifier found in the GDX file. The keys are case sensitive. The itemName is a long text which is typically shown to the user. The elements found between the <sel> …</sel> tags can be used as filters in table definitions, or interactively by the user.

A specific tag is <aggreg>yes</aggreg>. When found for an item in the rows, it will be shown twice in the table: once in the top part, and then again.

Includes

The XML processor allows to use includes, as seen below:

```
<xi:include href="generated.xml">
<xi:fallback>
   <xi:include href="generated_example.xml"/>
</xi:fallback>
</xi:include>
```

These includes can e.g. be generated by a GAMS process to pass run specific item lists to the exploitation tools. The following code from "GTAPinGAMS" project shows how to may look like:

```
loop(i,
     put "<activity>" /;
     put "    <itemName>", i.te(i),"</itemName>" /;
     put "    <key>", i.tl,"</key>" /;
     put "    <sel>all,sectors</sel>" /;
     put "</activity>" /;
);
```

That allows to group view definitions used by several tasks in one file, and to use these groups for a specific task. The following shows an example from CAPRI:

```xml
<?xml version="1.0"?>
<tables  xmlns:xi="http://www.w3.org/2001/XInclude">
<themes>Welfare,Farm,Farm - totals,Feed,Fertilizer,Environment</themes>

<xi:include href="views\coco_tables.xml"/>
<xi:include href="views\capreg_tables.xml"/>
<xi:include href="views\dimdefs.xml"/>

</tables>
```

The used of <xi:include> allows to also generated XML files on the fly by GAMS and pass it to the viewer, e.g. to reflect flexible lists of regions.

The structure of the GAMS generated gdx files used by the exploitation tools

The exploitation tools load directly the gdx-files generated by the GAMS processes linked to the tasks described above. The gdx-files only store non-zero numerical values. The main content of a gdx file are two types of records. The first type provides a list of all labels used to identify the numerical data in the gdx file as GAMS does not support numerical indices, but requires character labels. The list does not distinguish for which data dimensions the labels are used. They are hence typically a mix of product, activity, region and further labels. The second type of records belongs to GAMS parameters (scalars, vectors, or multi-dimensional tables). Each non-zero numerical item in each parameter has its own record. Each of these records provides the numerical data in double precision (depending on the parameter type there may be different data stored in one record, as for variables its upper and lower bound, current level and marginal value etc.), and a vector of indices pointing in the list of codes described above.

Loading the data from gdx files

The data matrices generated by the different tasks as described above and stored in gdx-files are typically rather sparse, so that it seemed appropriate to load the data from the gdx-file into hash tables for exploitation purposes. That is done in a two step procedure. In the first step, all

records from the gdx file are read and vectors of all found indices are stored. The length of each data dimension is only known when all data records are read, and is equal to the number of unique indices for each dimension. Once all records are read, the final length of these index vectors then defines a linear index room for the multi-dimensional table. In a second step, the records are read again, and the index vectors for each record now allow to define a linear index in the total table. A hash code is derived from that linear index to store the numerical values into a hash table. As the number of items to store in the hash table is known beforehand, a rather simple hash table implementation can be used. If necessary, step one can be run over several parameters which may be hosted in several gdx files, so that results from different runs can be merged into one hash table.

As the gdx-files provide lists of all labels used in any parameters stored in that gdx-file, the index vectors allows to build lists of labels linked for each index in a data dimension. There exists an additional storage type in the gdx-files to retrieve long-texts to the labels as defined in GAMS set definitions. However, one label may occur in different sets with different long texts, and the gdx-file does not store a possibly user defined relation between a data dimension of a parameter and a specific set, an option termed domain checking in GAMS. In order to link hence long-texts to the labels used for a specific data dimension, two options are possible. Firstly, at run time the user may interactively re-establish the link between data dimensions and specific sets, and thus add long-texts to the labels used on that data dimension based on his knowledge. Or the relation may be hard coded in the JAVA code.

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 a 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 include file(s) with run specific settings and that the names of these files are passed via the include file to GAMS.

Scenario editor

The scenario editor is an optional tool to be embedded in a GGIG user interface which supports the user in setting up run specific include files where the content is *not* stemming from GUI controls. That parallel way to define run specific input is typically necessary for more complex tools where e.g. policy scenarios are defined in GAMS code.

The scenario editor is a "predefined" task which must be named "Define scenario", e.g.

```
<task>
        <name>Define scenario</name>
        <userLevls>runner,Administrator,developper,debugger</userLevls>
</task>
```

A related setting stores the directory where the input files are found:

```
<scenarioDir><attr>scen</attr></scenarioDir>
```

The default location for user edited files is a sub-directory under scen called "user_scenarios". That location can be overwritten by the following setting:

```
<userScenarioDir><attr>..\gams\scen\userScenarios</attr></userScenarioDir>
```

The tag "calledBy" in the fie header of generated file is filled if the following field is set:

```
<callingFileForScenarioEditor><attr>COM_.gms</attr></callingFileForScenarioEditor>
```

Batch execution

The batch execution facility is a tool which:

- Allows executing many different tasks after each other without requiring user input.

- Reports the settings used, any errors and GAMS result codes in a HTML page from which they may queried at a later time.

- Ensures that each new run generates its own listing file, which can be opened from the HTML page.

- Allows storing the output of the different runs in a separate directory, while reading input from unchanged result directories.

The purpose of the batch execution facility is therefore at least twofold. On the one hand, it allows setting up test suits for the GAMS code of a project such as checking for compilation without errors for all tasks and different settings such as with and without market parts etc. Secondly, production runs of e.g. different scenarios can be started automatically. Timer facilities allow starting the batch execution at a pre-scheduled time. Along with functionalities to compare in a more or less automated way differences in results between versions, the batch facility is one important step towards quality control.

Each generated include file comprises a block of lines, commented out, which can be copied into a text file and used with the batch file facility. For details on the use of the batch execution utility, refer to the user guide.

Generate GAMS documentation

The GUI comprises a tool to generate for each GAMS file and each symbol used HTML pages which are interlinked. For details on the code documentation facility see the technical document "Javadoc like technical documentation for CAPRI" to be found on the Capri web page under technical documents.

The controls on top allow the user:

- To define in which directory the "EXP", "REF" and "GDX" files are stored which serve as input into the documentation generator.

- To choose the directory where the HTML files will be generated.

- To select the tasks covered by the documentation generator.

For details on the use of the GAMS documentation utility, refer to the user guide

Background

System such as CAPRI have grown over years to a rather complex (bio-)economic modelling system. Its code base consists of hundredth of single GAMS files, and ten thousands of lines. Not only newcomers face the challenge to get an overview about dependencies in the huge code base and to link the technical implementation to methodological concepts and documentation. On top, the large-scale character of CAPRI often asked for technical features in the GAMS code which are far from the solution chosen for tiny examples as often presented in courses, as the wide spread usage of dynamic sets, conditional includes, the usage of $batcinludes or the application of the grid solve feature.

The task of documenting and keeping an overview of the CAPRI code base is certainly not eased by the fact that basically any object in GAMS has global scope. The concept of functions of subroutines underlying many other programming languages with clearly defined lists of variables passed in and out is not implemented in GAMS. Encapsulation and modularisation are hence not naturally supported by GAMS. That also renders automated documentation of the code more challenging compared to other languages.

Since quite a while, CAPRI user community discusses about some refactoring of the code base on more clearer coding standards with the aim to ease code maintenance, documentation and further development. That refactoring should also cover standard for in-line documentation, including a better link to the methodological documentation. The project CAPRI-RD (2009-2013) has attacked some of these tasks for CAPRI in specific working packages. But clearly, that will only become success if the underlying concept is generally accepted and implemented by the community of CAPRI developers. That means that the value added of following coding and documentation standards must be visible to any developer.

The section here shows how to generate an easy to maintain and useful technical documentation for GAMS based projects such as CAPRI, based on the example of JAVADOC (http://de.wikipedia.org/wiki/Javadoc). It is organized as follows. The next short paragraphs will list desired properties of a technical documentation for a system such as CAPRI, followed by a more detailed discussion of a proposal for an implementation which is working as a prototype. The last chapter will then show selected screenshots.

Desired properties

The main properties an automated technical documentation of a tool under GGIG such as CAPRI should fulfil are as follows:

- Avoiding redundancies, i.e. information should whenever possible only inputted once. Specifically, existing in-line documentation already present in the code should be ported over to the technical documentation automatically.

- Changes in the code structure should possibly be reflected automatically

- The documentation must be able to reflect different tasks projects and to differentiate between instances of the same GAMS project used in different configurations (e.g. for calibration or simulation)

- Its biggest part of the technical documentation should be constructed directly from the code base in an automated way.

- It should also collect information from the SVN versioning system

Technical implementation

The main ingredients of the implementation are as follows:

- The final format of the technical documentation is based on automatically generated static HTML pages, following the example of JAVADOC, with some JavaScript to allow for collapsible trees

- The methodological documentation of a project should be edited in Word, and converted into a PFD-document. It will comprise references to GAMS sources (individual GAMS files) or even GAMS objects (variables, equations, models, parameters). Those references can be addressed in the GAMS code, and the HTML pages will allow opening the PDF-document at the referenced point.

- As with JAVADOC, technical documentation should be edited as in-line comments into the GAMS sources, based on clear in-line documentation standards. Each GAMS source as a file header with standard properties about the file.

- In-line documentation will be mostly based on two levels: the level of individual GAMS files and on the level of individual GAMS objects. In some cases, that may

require to break down larger programs in smaller pieces, with a clear task and eventually clear inputs and outputs.

Overview

The following diagram depicts the general approach. The SVN server will host the GAMS sources, the documentation builder (Builddoc) as a Java application and the PDF with the methodological documentation. Users synchronize their local work copies with the server. In order to avoid developing in Java a new parser for GAMS code, the GAMS compiler itself is used to generate the necessary input for the technical documentation. Two different types of files for each "project" or "instance" included in the documentation are used for that purpose so far:

1. So called **"REF" files**, which list information in which files and in which line symbols are declared, defined, assigned and referenced. They also comprise information about long texts and domain of the symbols. The "REF" file can be generated by the argument rf=*filename* when GAMS is called (e.g. "GAMS capmod a=c rf=capmod.ref"). As the GAMS compiler itself is used, conditional includes and the like are automatically treated as during execution time. That opens also the possibility to include the generation of the documentation in the GUI.

2. **GDX files generated with an empty symbol list at compile time** ($GDOUT module.gdx; $UNLOAD; $GDOUT). The resulting GDX file will comprise all sets, parameters etc. used by the programs, and most importantly, the set elements as declared. The name of the GDX file could be passed as a parameter by the GUI.

Those files hence reflect the actual local code base with any local modification, and can be generated for a specific instance of each GAMS project (e.g. in case of CAPRI's simulation engine, CAPMOD, with and without the market module etc.). A JAVA application named *Builddoc* parses both types of files, on demand for several projects, and generates static HTML pages. The GAMS code comprises in-line comments carrying information about references to the methodological documentation, and the HTML pages comprise calls to the editor to open the actual source code at the local machine, as well as information about relation between the different GAMS Symbols.
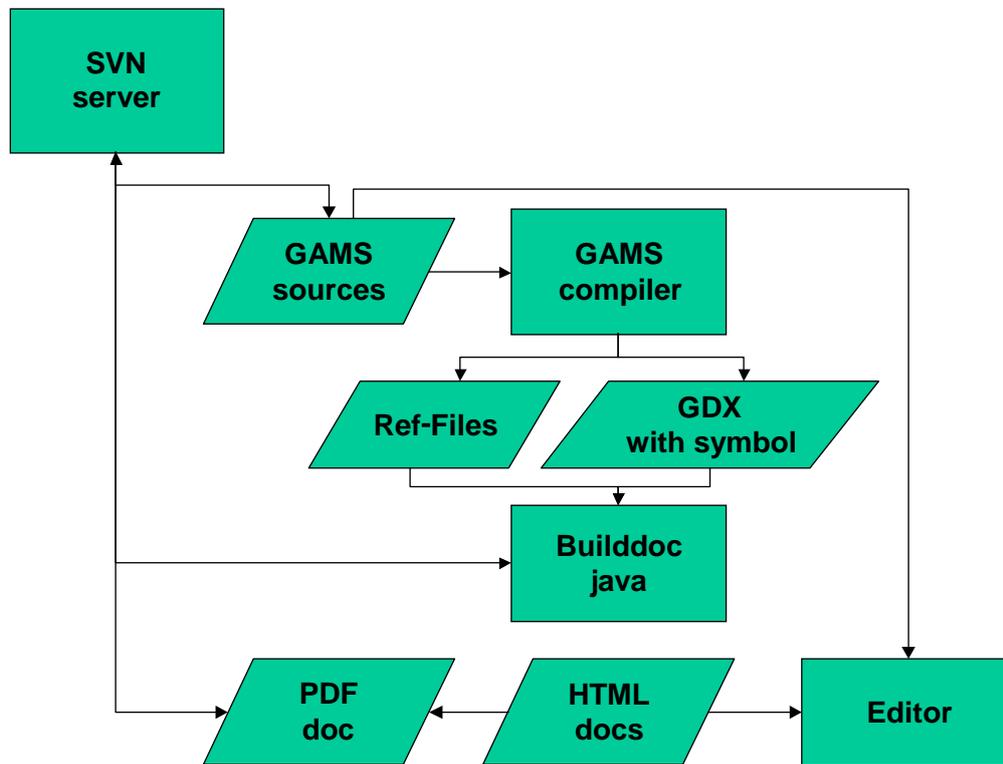
Handling of GDX files

The "expand" option generates information about GDXIN and GDXOUT statements as those are executed at compile time. Consequently, files addressed via GDXIN or GDXOUT are automatically reported in the documentation system.

Hovever, the file does not comprise information about the "execute_load" and "execute_unload" executed at run time. That is quite clear, as the statements may be comprised in program structures as loops or if statements where there are never reached at execution time. We need hence a work around to report those files in the documentation system if we would avoid writing a new GAMS parser.

However, "$IF EXIST" statements are taken into account by the expand command. It is therefore proposed to put an "$IF NOT EXIST" combined with an abort statement before all "execute_load" statements. As seen in:

```
$IF NOT EXIST ..\dat\arm\allpop.gdx $ABORT ..\dat\arm\allpop.gdx is missing
  execute_load '..\dat\arm\allpop.gdx' WorldPop;
```

By doing so, the program will already at compile exit if one of the necessary files is missing. That avoids starting a process and eventually overwriting files which then will stop later due to missing input data. The HTML page will report that sequence as:

Includes found in GAMS\CAPTRD.GMS :		

Top | Declarations | Definitions | Assignments | References | Includes | Is included by

File	Project (s)	Action
DAT\ARM\ALLPOP.GDX	captrd	IF EXIST - next Line : ' execute_load '..\dat\arm\allpop.gdx' WorldPop;'

The use of "$IF EXIST" in the context of "execute_unload" can only be motivated with the fact to produce code which is better documented. Here, is it proposed to warn the user at run time about the fact that the file is overwritten.

Index

References

Britz W., and M'Barek R. (2003). BenImpact: a decision support system for agricultural policy in Benin. Poster paper at 25th International Conference of IAAE, Durban, South Africa 16-22 August

Britz, W. (2014a): A New Graphical User Interface Generator for Economic Models and its Comparison to Existing Approaches, German Journal of Agricultural Economics 63(4): 271-285

Britz, W. (2014b): The Graphical User Interface for CAPRI version 2014, Uni Bonn, Institute for Food and Resource Economics, http://www.capri-model.org/docs/Gui2014.pdf

Britz, W. and Kallrath, J. (2012). Economic Simulation Models in Agricultural Economics: The Current and Possible Future Role of Algebraic Modelling Languages, in: Kallrath, J. (eds.). Algebraic Modelling Systems: Modelling and Soving Real World Optimization Problems, Springer, Heidelberg, Germany, 199-212

Britz, W., Dees, M., Walkiewicz, J. (2014): A Forest Sector Model for the Region Baden-Württemberg in Germany, selected paper presented at the Bioenergy from Forest 2014, September 15-18, 2014, Helsinki (Finland)

Britz, W., Perez Dominguez, I. and Badri Narayanan, G. (2015): Analyzing results from agricultural large-scale Economic Simulation Model: Recent Progress and the Way Ahead, German Journal of Agricultural Economics, Forthcoming

Britz, Wolfgang (2009). Sequentially linking a village CGE to farm-household models - a focus on transaction costs. Presentation at the workshop on "Evaluation of Rural Development Policies: Theory and Application", University of Kiel. July 13-14, 2009.

Dol, W. (2006). GAMS Simulation Environment. LEI The Hague, 128 pages, URL: http://www3.lei.wur.nl/gamstools/gse.doc

Heidecke, C., Heckelei, T. (2010). Impacts of changing water inflow distributions on irrigation and farm income along the Drâa River in Morocco, Agricultural Economics 41(2): 135 – 149

Jansen, J., Adelle, C., Crimi, J., Dick, J., Helming, K., Jacob, K., Janssen, S., Jordan, A., Podhora, A., Reis, S., Roosenschoon, O., Saarela, S.R,. Söderman, T., Turnpenny, J., Weiland, S. and Wien J.E. (2012): The LIAISE Approach to Unite Researchers and Practitioners in a Community of Experts on Impact Assessment. 2012 Berlin Conference on the Human Dimensions of Global Environmental Change on "Evidence for Sustainable Development". In: http://www.liaise-noe.eu/system/files/Berlin%20conference_LIAISE%20approach_final.pdf.

Kuhn, A., Britz, W., Willy, D. K., van Oel, P. (2014): Simulating the viability of water institutions under volatile rainfall conditions – The case of the Lake Naivasha Basin, Environmental Modelling & Software, available inline since 16th September 2014

Lengers, B., Britz, W., (2012). The choice of emission indicators in environmental policy design: an analysis of GHG abatement in different dairy farms based on a bio-economic model approach, Review of Agricultural and Environmental Studies 93, 117-144

Nilsson, M., Jordan, A., Turnpenny, J., Hertin, J., Nykvist, T. and Russel, D. (2008): The use and non-use of policy appraisal tools in public policy making: an analysis of three European countries and the European Union. Political Science, 41 (4): 335–355

Rizzoli, A.E. et al. (2009). Updated version of final design and of the architecture of SEAMLESS-IF Report No.47, SEAMLESS integrated project, EU 6th Framework Programme, contract no. 010036-2, www.SEAMLESS-IP.org, 31 pp, ISBN no. 978-90-8585-590-3

Schroeder, L. A., Gocht, A., Britz, W. (2014): The Impact of Pillar II Funding: Validation from a Modelling and Evaluation Perspective, Journal of Agricultural Economics (in press)

Wieck, C., Schlüter, S. W., Britz, W. (2012). Assessment of the Impact of Avian Influenza Related Regulatory Policies on Poultry Meat Trade and Welfare, The World Economy 35(8): 1037-1052