Scenario definition via the GUI in CAPRI – mission impossible?

- Wolfgang Britz, April 2009 –

Background

The way policies are modelled in CAPRI is rather close to the actual policy instruments: sales quota for milk, the A-B-C sugar regime, set-aside obligations, the different premium schemes, tariff-rate quotas, preferential access, market intervention or export subsidies are implemented more or less directly, and not as price wedges as e.g. in CGEs. As a consequence, changing the instruments in scenarios requires advanced knowledge about the internal data structures and statements in the CAPRI GAMS code handling these instruments. The ever increasing complexity of the CAP, and different projects adding new instruments such as the entry price systems for fruits & vegetables or the details of the EU sugar CMO render it even for experienced CAPRI modellers hard to change these policies.

The impossibility to easily change policies in a scenario provides a strong entrance barrier to CAPRI which prevents to a certain extent naïve and mis-guiding applications of the system. In that context, the standard version of GTAP is always mentioned as a bad example for a model with a low entrance cost where inappropriate model use might destroy the model's trade mark. The GTAP community has reacted to that by instruments as a peer reviewed working paper series. The dis-advantage of the current CAPRI solution is equally clear: only a handful of advanced users is able to implement new scenarios, and important clients such as JRC-IPTS with a focus on application rather than on development are discouraged from developing and analysing own scenario analysis.

Since the very first version of CAPRI there as always a discussion about the if and how of a scenario editor. Indeed, the very first C-FORTRAN based CAPRI GUI features an editor to define the then active coupled premiums of the CAP. However, it quickly turned out that the maintenance costs of the solution where simply to high and the instrument not flexible enough. Since then, no new editor was developed. The question of an editor came up again in

the context of SEAMLESS. Here, for some selected policy elements a policy editor was developed.

Generally, the following options for a CAPRI policy editor could be imagined.

1. Firstly, no editor at all, as it is currently the case. Clearly, no further coding is required, and the maintenance costs are low. However, there is the real danger that almost nobody will be able to implement scenarios after further complications of the CAP and implementation of Pillar 2 instruments in the context of the CAPRI-RD project. The envisaged refactoring of the CAPRI GAMS code base in CAPRI-RD might with a hopefully clearer code structure contribute to a somewhat more user friendly implementation of policies. It is however not very probable that the solutions in the GAMS code will be “self-explaining”. It is not only necessary to know where the instruments are defined, but also how they mapped into data structures, which labels are used, which values are allowed etc.
2. The second option builds on the exploitation possibilities of the GUI, basically, it would open the tabular views for editing. The advantage would clearly consist in the fact that a major part of the existing Java code base could be used, and the user would face an already familiar instrument. Technically, some limited changes in the underlying data structures would be necessary. It remains to be seen if that option could be used in parts. But it would requires some serious re-factoring in the GAMS code to map the different data structures storing e.g. the premiums or trade policy instruments into one data cube.
3. Thirdly, a new generic editor might be developed, e.g. building on the experience with SEAMLESS. That might be inviting as e.g. upper and lower limits on the allowed ranges might be integrated. However, it is questionable if it is possible to develop with limited resources such an editor which is so generic that the introduction of new policy instruments could be handled without further coding. Given limited resources for further Java development, and the need to maintain and update the code in the future renders that solution not really viable.
4. The last option is a very limited expansion of the current idea to have all scenario relevant code assembled in one GAMS file, only. Instead, logical parts of a scenarios and variants of such parts relating e.g. to premiums, administrative prices, border policy and the like would be stored in small separate files. These code snippets may then be changed by the user and can then be combined into more complex scenarios.

The solution proposed

Currently, scenario definition files typically already comprise includes where code snippets relating to certain policy instruments or other assumptions are stored. A typical example is a small file with time series of the milk quota quantities used in a scenario. It seems therefore rather straightforward to store certain policy instruments, such as premium schemes, quotas or set-aside in small separate files so that they might be combined by the user to new scenarios – say, the decoupled payments from the MTR with a certain set-aside rate and certain milk quotas.

One way to proceed is to prepare small files which define alternatives for certain policy instruments, such as different levels of decoupling or implementation of the decoupled premiums. Those code snippets would be under version control in the standard distribution of CAPRI. The user could “assemble” a scenario from existing pieces, or alternative, modify them and store them into an own scenario file. That would also render the code more transparent, as it would be clear where the relevant snippets for certain instruments are stored, and where and how they are implemented. It would also cut almost any connection between the GUI and the GAMS code: the GUI would only need to know where the snippets are stored.

The user has two options to work with the snippets:

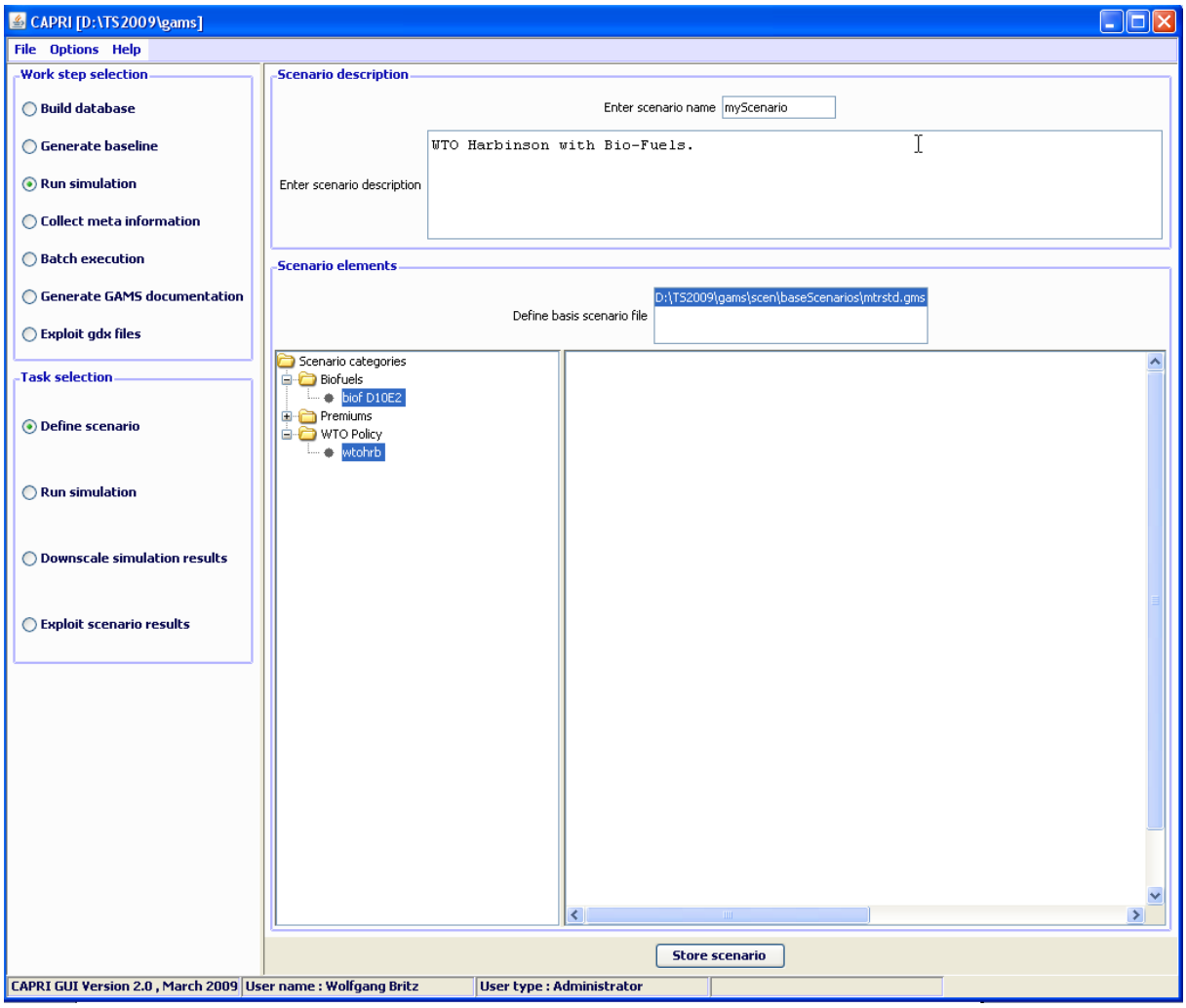
1. The existing snippets can be combined to a new scenario. The scenario file generated will then comprise an $include pointing to the snippet.
2. Any existing snippets can be changed via the GUI. The full code of the changed snippets becomes then part of the generate scenario file.

In order to test the proposal, a new directory “Scen” comprising sub-directories relating to specific policies instruments or other exogenous assumptions has been added to the trunk. Sub-directories may comprise code-snippets stored as small GAMS files and/or sub-directories. The user might then select from each of the categories one or several “code snippets”, set a name for his scenario name and add a description. He can also double-click on a snippet and then edit it.

The obvious advantage of the approach is that would allow e.g. for a training session to pre-define elements of scenarios which the user can be combine on its own. The connection between the GAMS code and the interface is very limited: adding new elements to scenarios would not require any modification to the Java code. The users can inspect the small snippets

and learn from them how these policies are defined in CAPRI, and might easily generate new variant from the existing code.

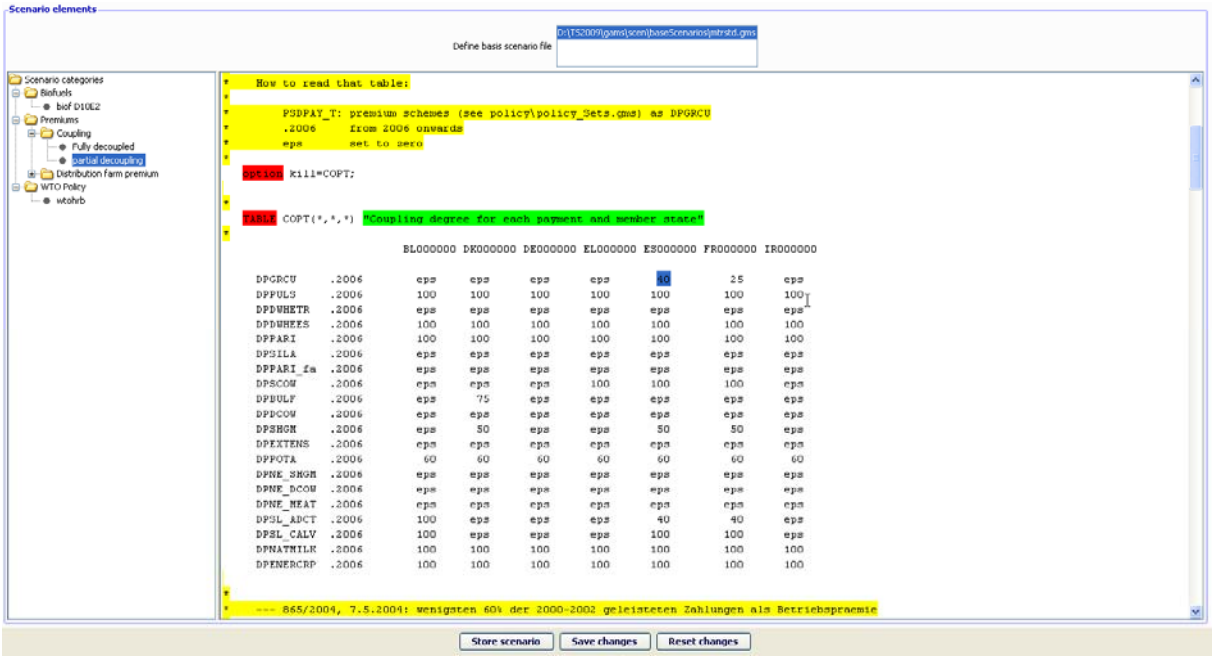
Functionality of the "Define scenario"



A new task "Define scenario" had been added to the work step "Run scenario". Chosing the task adds the panel with GUI elements shown above. The panel consist of two major panes:

1. A top pane where the user can enter a name for his new scenario, and a description text.
2. A bottom pane where the user can define the base scenario to start with (currently in the trunk "MTDSTD.gms") and the snippet to add. The available snippets and their structure is shown on the left hand side in a expandable tree which shows the sub-directories found under "gams\scen", with the exclusion of a sub-directory called "baseScenarios" and the ".svn" directories. Empty directories are not shown. The user may select any number of snippets, even several from the same sub-directory. Double-clicking on one of the snippets shows the content of the file on the right hand side, so

that the user can inspect the code as seen below in more detail. GAMS keywords are shown in red, comments in yellow and strings in green. He can also edit the file – changes are shown in blue. Once changes had been saved, the tree shows a (user modified) behind the category. The user can also remove the changes from snippets.



Storing the scenario then generates a file as shown below, user name, the reference to CAPMOD.GMS and the date and time are automatically added by the GUI. The files will be added to the files stored in “gams\pol_input”.

```
*******************************************************************************
*
*   Author:      Wolfgang Britz
*   Called by:   CAPMOD.GMS
*   Created:     09-04-2009 14:42:34
*   Purpose:     Scenario definition file
*
*******************************************************************************
*
* User supplied description :
*
* Biofuels, fully decoupling and WTO Harbinson proposal combined.
*******************************************************************************
*
*  Baseline scenario
*
$include mtrstd.gms
*
*  Category : WTO Policy
*
$include scen\WTO_Policy\wtohrb.gms
*
*  Category : Biofuels
*
$include scen\Biofuels\biof_D10E2.gms
*
*  Category : Coupling
*
$include scen\Premiums\Coupling\Fully_decoupled.gms
```

Consequences for the GAMS code

In order for the solution to work, the GAMS code must be defined such that all scenario elements can be included at the same position in the GAMS code.