GEM

Information paper

6 June 2012

Contents

Tables

Figures

Executive summary

Notes for future of this document:

- For now, this doc is simply called GEM.doc and it resides in Phil's GEM root directory.
- Morph this into a general EMI "installation and high-level user manual" document with chapters or sections for each model in EMI.
- Make the installation of EMI and EMI models section quite comprehensive, coherent and standalone. The chapters on individual models should remain fairly high-level, and cover how to use the models as standalone models as well as within EMI. But refer to additional documents, where appropriate and if such docs exist, to get into the detail of data descriptions, model formulations etc.
- An "EMI" tab should be added next to the "vSPD data" tab on the portal for all things related to EMI and EMI models.

1 Introduction

1.1 These notes present some basic information about installing and operating GEM, the Electricity Authority's generation expansion model. At this stage, the notes assume that GEM is being used independently of EMI, the Windows-based graphical interface to GEM and other Authority models.

2 Installation

GAMS

2.1 GEM requires a licensed version of the GAMS software to be installed. Prior to installing the GAMS software, users must contact the GAMS Development Corporation in Washington DC to arrange for the purchase of a GAMS license file.[1] If users plan to modify GEM by adding new variables or equations to the model, then a developer license will be necessary. If users plan to simply modify the data inputs and use GEM 'as is', then a runtime license will suffice. A runtime license is less expensive than a developer license, and can be subsequently upgraded to a developer license if required (for a total cost equal to the cost of a developer license).

2.2 While the GAMS code for GEM is portable across operating systems, it is assumed throughout this document that a Windows platform is being used.

2.3 The latest version of GAMS should be obtained from http://www.gams.com/download/ and installed in the usual manner as for any other Windows application. The license file, a small text file, must first be saved to the computer on which GAMS is being installed, as it will be required towards the end of the installation process.

2.4 Once GAMS is installed and working, it is advisable to add the GAMS system directory to the path environment variable. The GAMS system directory is the location where GAMS was installed. Ask your IT administrator if you don't have the rights to do this or you don't know how.

GEM

2.5 Create a directory for your GEM files. Throughout this document it is assumed the GEM directory will be `c:\GEM\`, although it can have any name and be located anywhere. However, it is highly advisable that you don't put your GEM directory in your GAMS system directory or in any other location where programs or applications reside. All GEM files, whether GAMS code or input data, should be managed in the same way as any other collection of data files would be managed.

2.6 Within the GEM directory, create three subdirectories and name them `Data`, `Output`, and `Programs`, respectively.

2.7 The latest version of GEM is available from http://code.google.com/p/gem/downloads/list as two zip files. Download both files, the data and the codes, and unpack them. Place the entire contents of the data file in the data directory, `c:\GEM\Data\`, and the entire contents of the codes file in the programs directory, `c:\GEM\Programs\`.

2.8 The GEM data directory will contain a collection of GDX files. These are the GEM input data files. The GEM programs directory will contain a collection of text files with the suffix .gms, .inc, or .txt. There will also be a binary file called `GEMdeclarations.g00` in the GEM programs directory. Whenever GEM is invoked, or executed, the GEM output will be placed in an automatically created subdirectory inside of the GEM output directory.

[1] See http://www.gams.com/contacti.htm.

3 The GEM components

3.1 At the most fundamental level, GEM is comprised of some GAMS code and a few GDX files containing the input data. When GEM is invoked, the GAMS code is compiled, the input data is imported, the current configuration of the model is solved, and a group of output files are created.

3.2 Very importantly, however, the logic of the model, which is embodied in the GAMS code, is completely distinct from the input data. Thus, two completely different input data sets can be compiled, and GEM can be invoked twice – once for each data set. Each run of the model would make use of the exact same GAMS code, i.e. the logical structure and mathematical formulation of the model is the same for both cases – but the model runs would be very different, and would yield very different outcomes or solutions, by virtue of the two different data sets.

GAMS code

3.3 The GAMS code that comprises GEM is split into four main files:

(a) `GEMdeclarations.gms` – declares all of the symbols (sets, scalars, parameters, variables, equations and files) used throughout GEM. In addition, the equations are initialised in `GEMdeclarations.gms`, which means the mathematical formulation of the model is specified. No data is imported and no data manipulations or computations are undertaken within `GEMdeclarations.gms`;

(b) `GEMdata.gms` – continues sequentially from `GEMdeclarations.gms` via the GAMS work file, `GEMdeclarations.g00`. The following tasks are undertaken by `GEMdata.gms`:

(i) the input data is loaded from the specified GDX files;

(ii) integrity checks are performed on the imported or loaded data;

(iii) various data manipulations and computations are completed to prepare the data so that the model can be solved; and

(iv) a range of input data summary files are produced.

(c) `GEMsolve.gms` – continues sequentially from `GEMdata.gms`, solves the model(s) and collects and saves the output into GDX files for use by `GEMreports.gms`. Depending on the particular run configuration, many versions of GEM may be solved; and

(d) `GEMreports.gms` – reads in the GDX files created by `GEMsolve.gms` from one or more prior GEM runs, and produces a collection of reports. Some of these reports are in the form of CSV files that may be used in other applications to produce plots of the output.

3.4 While it is certainly helpful to know how to edit GAMS code, users are able to operate GEM without delving inside any of the above four files. A meaningful GEM run can be accomplished simply by editing the input data files and/or the various include (.inc) files. The .inc files are discussed later in section 5.

3.5 There are additional .gms files to be found in the GEM programs directory besides the four noted above. However, they fall into the miscellaneous or utility category and will be described later in this document. For now, it is sufficient to understand that the GEM model is essentially comprised of the above four GAMS programs or codes.

3.6 Users with no familiarity with GAMS or with no intention of editing the GEM codes would nonetheless most likely find it beneficial to read through `GEMdeclarations.gms`. The description and, where relevant, the units given to each of the symbols will dramatically enhance users' understanding of GEM. Viewing the input data GDX files is also likely to be a more useful exercise after first reading `GEMdeclarations.gms`.

3.7 Finally, the mathematical specification of the model can be precisely understood by reading `GEMdeclarations.gms` for users with some GAMS experience. But even users without such experience can, with a little effort, obtain the essence of each constraint.

Input data

3.8 A GEM input data set is comprised of three, or possibly four, GDX files. Although having said that, it is reasonable to compile and maintain more than one version of each or the three (or four) GDX files.

3.9 The three files are the so-called standard input data, regional and network data, and energy demand data. For example, an input data set based on a 9-block load duration curve and a 2-region network might be the following three GDX files:

(a) `standardGEMinput9LB.gdx`;

(b) `2RegionNetwork.gdx`; and

(c) `NRG_2Region9LB_Standard.gdx`.

3.10 These three files, and others, are available in the zip data file available for download from http://code.google.com/p/gem/.

3.11 Taken together, these three files contain 133 data symbols or data structures, and comprise a complete input data set for GEM. While the files can be edited and the data values for each symbol changed, the data structures themselves should not be modified without making commensurate modifications to the relevant GAMS code.[2] In other words, the symbol names and domains must remain as declared in `GEMdeclarations.gms`.

3.12 The standard input data file, e.g. `standardGEMinput9LB.gdx`, contains all of the input data (114 symbols) that is invariant with respect to the network or regional configuration. Conversely, the regional and network data file, e.g. `2RegionNetwork.gdx`, contains all of the input data (18 symbols) that depends on the particular network or regional configuration. The energy demand file, e.g. `NRG_2Region9LB_Standard.gdx`, contains a single data symbol, which, as its name suggests, contains the energy demand by region and load block.

3.13 The regional configuration in GEM goes hand-in-hand with the network configuration. Stated differently, the notion of a node on the network is synonymous with a region in GEM. Thus, a 2-region data set implies a 2-node network.

3.14 One way to change the underlying assumptions in GEM is to create a new input data set, i.e. create new versions (presumably with new names) of the standard input data, regional and network data, and energy demand data files. But this can be tedious if only a few data symbols require editing. An alternative method, although by no means the only one, is to make use of the override file, the fourth type of input GDX file. The override file is used to provide alternative values to GEM for one or more designated data symbols from the set of three input GDX files. The symbol name in the override file is the same as the symbol name being overridden, or overwritten, except that the letters `Ovrd`, for override, are appended to the end of the symbol name.

3.15 For example, the override file called `mds1_Override.gdx` contains four data symbols – `i_fuelPricesOvrd(f,y)`, `i_fuelQuantitiesOvrd(f,y)`, `i_co2taxOvrd(y)`, and `i_FixComYrOvrd(g)`. Respectively, these four symbols are used to override the value for `i_fuelPrices(f,y)`, `i_fuelQuantities(f,y)`, `i_co2tax(y)`, and `i_FixComYr(g)`,

[2] See section 4.

each of which can be found in the standard input data file. If GEM is instructed to use an override file, then when GEMdata.gms is invoked, it will load the data symbols from the designated override file, and use the data values therein to override the data values originally loaded for the commensurate symbols.

3.16 The data symbols in an override file must be declared and loaded in GEMdata.gms. To date, only the four noted above are so declared and loaded. At some point, the publicly available code will accommodate overriding of all data symbols and the symbol declarations will be moved to GEMdeclarations.gms.

3.17 Alternative methods of modifying the GEM input data will be described later in section 4.

Basic GEM configuration concepts

3.18 There are four so-called .inc files – GEMpathsAndFiles.inc, GEMsettings.inc, GEMstochastic.inc and GEMreportSettings.inc – that require editing by users in order to configure a GEM run. While the details of these files will be described later in section 5, the four .inc files – particularly GEMstochastic.inc – embody important conceptual ideas that are central to understanding how GEM operates. These concepts are now described.

Runs and run versions

3.19 The notion of a GEM run and a version of a run relates to a thematic collection of GEM jobs or invocations. For example, a GEM run might use a particular input data set, i.e. the three GDX files, but there might be five, say, versions of the run, each of which is associated with an override file. The five run versions would yield five GEM solutions, each of which is different only because of the input parameters whose values were modified in the override files.

3.20 But the aforementioned case is just one example of how the distinction between runs and run versions manifests itself. It may be the case that a user constructs a set of run versions by compiling a completely new input data set for each version of the run, i.e. modifying values in one or more of the three input GDX files. In other words, a GEM run need not imply the use of a single input data set. Conversely, there are many run configuration settings (yet to be described) that may result in the same input data set producing vastly different model results, thus giving rise to multiple run versions of the parent run.

3.21 The key point is that it's entirely up to the GEM user to design the run and its run versions. It is envisioned that a set of run versions would represent minor variations of model inputs relative to a base case run version. A run can be associated with only one run version, or it can be made up of many run versions.

3.22 Both the run name (runName) and the run version name (runVersionName) are specified by the user in GEMpathsAndFiles.inc. All output related to a GEM run and its associated run versions is stored in an automatically created subdirectory with the GEM output directory. The name of the automatically created subdirectory is the character string assigned by the user to runName.

3.23 In addition, both the run name and the name given to the associated run versions are used by GEM in assigning names to the output files. Thus, users can easily trawl through a collection of GEM output and quickly identify the output files associated with any particular version of a run.

3.24 Finally, the reporting module of GEM, GEMreports.gms, can be instructed to generate reports based on the solutions to one or more run versions of a run, and even run versions drawn from more than one run.

Scenarios

3.25 It might be tempting to think of run versions as nothing more than scenarios. That is, for each run version, one or more parameter values are changed to represent alternative views of the 'true' but unknown parameter values, at least unknown with absolute certainty. However, the term scenario is reserved for use in `GEMstochastic.inc` where precisely that interpretation of a scenario is implied. So, for example, one of the input parameters in GEM is an annual CO_2 tax. It is possible in GEM to establish alternative scenarios relating to the CO_2 tax – for instance, low, medium and high CO_2 tax scenarios might be specified. The specification of scenarios in GEM is accomplished by defining elements of the set called `scenarios` in `GEMstochastic.inc`.

3.26 Scenarios in GEM are therefore used to describe the various states of uncertainty or stochastic futures that relate to a particular input parameter. Scenarios relating to many input parameters can be specified. For example, a low, medium and high CO_2 tax might be specified along with five variations of peak demand and 10 settings for the price of gas in each modelled year.

3.27 It is important to appreciate, however, that as more scenarios are modelled, the dimensions of the GEM model get greater and the model becomes computationally more difficult and/or it will take longer to solve. As will eventually become clear, GEM can be solved as a strictly deterministic model, or scenarios can be grouped together and weighted so that GEM is solved stochastically.

Sets of scenarios

3.28 Scenarios may be grouped into coherent sets and the model solved repeatedly, once for each set of scenarios.[3] Such a set of scenarios is termed `scenarioSets` in `GEMstochastic.inc`. The process of mapping scenarios into sets is accomplished via the mapping set called `mapScenarios(scenarioSets,scenarios)`, which can also be found in `GEMstochastic.inc`. Note that a scenario may be used in, or mapped to, more than one scenario set. Also, a scenario set must contain at least one scenario.

Experiments

3.29 In the same way that scenarios can be grouped into sets of scenarios, sets of scenarios can be grouped into experiments. Scenario sets are mapped to experiments via the following three mapping sets in `GEMstochastic.inc`:

- `timingSolves(experiments,scenarioSets);`
- `reoptSolves(experiments,scenarioSets);` and
- `dispatchSolves(experiments,scenarioSets),`

3.30 which are all combined into a set called `allSolves(experiments,steps,scenarioSets).`

3.31 The mapping of scenario sets to experiments is further explained in the next section, which discusses the steps in the model solution process. Suffice to say, within a single run version of GEM, the model is solved as many times as their elements, or `experiments-steps-scenarioSets` combinations in the set `allSolves`.

3.32 Hence, the process of defining and parameterizing scenarios, and then grouping scenarios into scenario sets and ultimately scenario sets into experiments, provides a lot of scope for the user to configure and solve a large number of models in a single run version, where all such models, potentially, employ the same input data set.

[3] Note that coherence in this setting is defined by the modeller, and not by the authors of GEM.

Steps in the model solution process

3.33 The concept of `steps` in GEM essentially refers to the distinction between solving the fundamental capacity expansion problem, i.e. what, where and when to build, and solving a simulation problem where the dispatch pattern associated with a given capacity expansion plan is simulated under various hydrology scenarios. GEM has three hard-coded steps:[4]

- `timing` – solves the investment timing problem, i.e. determines what new generation and/or transmission investment should be built, where it should be built, and when it should be built;
- `reopt` – solves a re-optimised investment timing problem where all but the peaking plant from the capacity expansion plan in the preceding timing problem is fixed, but a tighter set of hydrological conditions, i.e. a dry year, is used relative to the initial timing problem; and
- `dispatch` – solves for the dispatch only using a fixed capacity expansion plan. Generally speaking, the capacity expansion plan would be determined in the same run version using the `timing` and, possibly, the `reopt` steps. However, it is also possible to supply GEM with a capacity expansion plan and bypass the `timing` and `reopt` steps. In a typical run, or run version, the `dispatch` step is solved repeatedly using a variety of hydrological conditions.

3.34 The `timing` and `reopt` steps are often referred to as the integer or MIP solves whereas the `dispatch` step might be referred to as the linear or simulation solve.

3.35 The statement instructing GEM to solve the model is inside a nested loop structure. The outer loop is on `experiments`, the middle loop is on `steps`, and the inner loop is on `scenarioSets`. At any stage during the solution process, a single (`experiments`,`steps`,`scenarioSets`)-tuple will be active. All scenarios associated with the active tuple, i.e. via the previously discussed scenario mapping sets, will be simultaneously solved over. Readers familiar with mathematical programming models will appreciate that this then implies a subset of variables and equations in the model must have `scenarios` in their domain.

Historical hydro years and hydro sequence types

3.36 GEM is a long-term planning model so, quite naturally, it is solved for some user-specified number of years into the future, e.g. from 2012 through to 2045, say. The sequence of years over which GEM is solved is referred to as the modelled years and is denoted with a set called `y`. But GEM also makes reference to another set of years – the so-called set of historical hydro years, denoted `hY`. Historical hydro years in GEM refer to the historical years for which observed hydrological inflow data have been collated. The set `hY` contains a continuous sequence of years beginning in 1932.

3.37 The concept of scenarios is used to enable the user to specify the hydrological conditions to be assumed for any given solve of GEM. Needless to say, the options available to the user are many and varied so the mapping of historical hydro years to scenarios is quite complicated. In essence, the user needs to somehow specify one or more historical hydro years to be associated with each modelled year.

3.38 The use of scenarios to map historical hydro years to scenarios, and thereby to modelled years, is really just a special case of the use of scenarios.

[4] While the elements of the set called `steps` are hard-coded and therefore unavailable for users to modify, it is not essential that a GEM run, or run version, be configured to use all three steps.

3.39 It may be the case that one historical hydro year is selected and assumed to prevail in all modelled years. Or perhaps a simple or weighted average of several or all historical hydro years is assumed to prevail in each modelled year. Alternatively, one historical hydro year can be assigned to one modelled year, another historical hydro year to the next modelled year and so on and so forth. Conceptually, the possibilities are endless, and users need to be very clear as to what they are assuming and the implications of those assumptions.

Hydro sequence types

3.40 In order to understand how historical hydro years are mapped to modelled years, it is first necessary to understand the concept of hydro sequence types. GEM contains a hard-coded set called `hydroSeqTypes` with just two elements – `same` and `sequential`. `GEMstochastic.gms` contains a mapping set called `mapSC_hydroSeqTypes(scenarios,hydroSeqTypes)`, which is used to map hydro sequence types to scenarios. The mapping of historical hydro years to modelled years takes place on-the-fly in `GEMsolve.gms` according to the settings established by the user in `GEMstochastic.gms`.

3.41 If a scenario is designated to have a hydro sequence type of `same`, it implies that the historical hydro year mapped to that particular scenario is assumed to prevail in all modelled years. Conversely, if the scenario is mapped to a sequence type of `sequential`, then the historical hydro year/modelled year series is assumed to develop, well, sequentially. More specifically, the historical hydro year/modelled year mapping arrangement begins by taking the particular historical hydro year mapped to the scenario and assigning it to the first modelled year. See also `mapSC_hY(scenarios,hY)` described in paragraph 3.43. The historical hydro year/modelled year mapping arrangement continues by taking the historical hydro year that appears in set `hY` immediately following the historical hydro year mapped to the scenario, and assigns it to the next modelled year. And so on and so forth.

3.42 The process continues from the first through to the last modelled year. If the historical hydro year mapped to the scenario is towards the end of the historical hydro year set, it is possible that the mapping of historical hydro years to modelled years will exhaust the historical hydro years before all modelled years are assigned a historical hydro year. In such cases, once the last historical hydro year has been mapped, the process just wraps around to the first historical hydro year and carries on in sequence. Typically there are fewer modelled years than there are historical hydro years.

Mapping historical hydro years

3.43 The mapping set called `mapSC_hY(scenarios,hY)` in `GEMstochastic.inc` is used to assign historical hydro years to scenarios, and thereby to the modelled years in that scenario. So, for example, if the 1932 historical hydro year, say, was assigned to a scenario, and that scenario was associated with a sequence type of `same`, then the 1932 historical hydro year would be assumed to prevail in all modelled years. On the other hand, if a sequence type of `sequential` was assumed for the scenario, then the 1932 historical hydro year would be mapped to the first modelled year – 2012, say. And the 1933 historical hydro year would be mapped to the 2013 modelled year, 1934 to 2014, and so and so forth until all modelled years have been assigned a historical hydro year.

3.44 If more than one historical hydro year is assigned to a scenario, then a simple average of all hydrological conditions for all of the mapped historical hydro years is automatically computed and assigned to each modelled year in the particular scenario.

3.45 Because scenarios are mapped to scenario sets, it is possible to assign weights to each scenario mapped to a scenario set via `weightScenariosBySet(scenarioSets,scenarios)` in `GEMstochastic.inc`. Thus, in the special case where a scenario is defined for the purpose of managing historical hydro year mappings, the careful and judicious use of `mapScenarios`, `mapSC_hY`, and `weightScenariosBySet` makes it possible to assign a weighted average of multiple historical hydro years to one or more modelled years.

3.46 Due to the complex nature of designing and specifying an arrangement for mapping historical hydro years to modelled years, the actual mapping, whether or not it was what was intended, is collected into a set called `mapHydroYearsToModelledYears` during the execution of `GEMsolve.gms`. It is written into one of the GDX output files and is available for inspection after the model run is complete.

4 Working with GDX files

4.1 Need to write this section up properly – it's is just sketched out for now

4.2 Lots of options for editing input data:

(a) The GAMS IDE can be used to view or inspect a GDX file before deciding upon edits. It can also be used to export selected or all symbols to Excel or CSV files.

(b) GAMS code – do you thing in GAMS or use a bit of GAMS code to get the GDX into Excel. And then back again after editing in Excel.

(c) dumpGDX and reloadGDX batch files to create .gms files.

(d) See gdxutils/pdf at Help|docs|tools on the GAMS IDE. In particular, GDXdump and GDXmerge. The GDXdump command will convert a GDX file to GAMS-compliant .gms (or text) file

(e) GDXdiff on the Utilities menu of the GAMS IDE

(f) copy.awk and replace.awk for simple text file operations

(g) Python utilities for operating directly on GDX files, e.g. modify, add or remove contents. This is the best option – grab the symbols you want to edit from a GDX file and convert them to a CSV file, edit them, and then put them back into the GDX file. Download GDXutilities.zip from http://code.google.com/p/getemi/downloads/list. Read the Readme.txt??? Download and install Python - http://www.python.org/getit/releases/2.6/. Recommend v2.6.6. Make sure Python and GAMS system directories are added to the Path environment variable. Create a new system/environment variable called PythonPath that points to the apifiles\GDX folder in the GAMS system directory. On my computer, I have (32-bit Python and 64-bit GAMS), Python 2.6.6 (r266:84297, Aug 24 2010, 18:46:32) [MSC v.1500 32 bit (Intel)] on win32; and GAMS build: 23.7.2 WEX 27052.27054 WEI x86_64/MS Windows. Get the python GDX scripts from https://github.com/geoffleyland/py-gdx. Feel free to contact Geoff Leyland at geoff.leyland@incremental.co.nz if you use the Python scripts and would like some additional capability.

(h) Of course, EMI is another option for editing GDX files.

5 Running GEM

5.1 This section sets out a very minimal set of instructions for configuring and running GEM in standalone mode, that is, without the use of EMI. It is assumed that a licensed copy of GAMS has

been installed on the user's computer and that the latest version of GEM has been downloaded from http://code.google.com/p/gem/ and unpacked, and is ready for use.

5.2 In order to operate GEM in standalone mode, it is necessary to have access to a reasonable text editor. The GAMS IDE has an adequate in-built text editor.

5.3 If GEM is to be invoked, and the .inc files edited, from within the GAMS IDE, then it is necessary to create a GAMS project. From the file menu of the IDE, select `Project` and then `New Project`. Create a GAMS project file in the GEM programs directory and call it `GEM`. A file called `GEM.gpr` will now exist in the GEM programs directory; do not delete it.

Prepare an input data set

5.4 Obtain, edit, or otherwise prepare the three GDX files that comprise a GEM input data set. In addition, prepare as many override GDX files as are required. It is possible that no override files will be required. The preceding sections of this document, and in particular section 4, may be of some assistance in preparing an input data set.

5.5 The GAMS codes called `GEM1dc.gms` and `GEMgdx.gms`, which should be located in the GEM programs directory, can be used to create a new energy demand GDX file. The codes themselves contain usage instructions and indications of the data required.

Edit GEMpathsAndfiles.inc

5.6 Using a text editor, open the file called `GEMpathsAndSettings.inc` in the GEM programs directory and assign values, or character strings, to the following `$setglobal` variables:

(a) `$setglobal runName` – give the run a name, e.g. `myrun`.

(b) `$setglobal runVersionName` – give the current version of the run a name, e.g. `mds5`.

(c) `$setglobal runVersionDesc` – give the current version of the run a long or descriptive name, e.g. `High gas discovery`. `runName`, `runVersionName` and `runVersionDesc` should be selected carefully and so as to be meaningful. They are used in the production of reports. Brevity, however, is a virtue with respect to `runVersionName`.

(d) `$setglobal runVersionRGB` – give the current version of the run a 3-digit code to represent the RGB colour mix. This ensures that plots and graphs are produced consistently.

(e) `$setglobal GEMinputGDX` – specify the name of the standard input GDX file to be used, e.g. `standardGEMinput9LB.gdx`.

(f) `$setglobal GEMnetworkGDX` – specify the name of the regional and network data GDX file to be used, e.g. `2RegionNetwork.gdx`.

(g) `$setglobal GEMdemandGDX` – specify the name of the energy demand GDX file to be used, e.g. `NRG_2Region9LB_TiwaiGoes.gdx`.

(h) `$setglobal GEMoverrideGDX` – if an override file is to be used, provide its name, e.g. `mds3_Override.gdx`.

(i) `$setglobal useOverrides` – indicate whether or not an override file is to be used; 0 denotes no and 1 denotes yes.

(j) `$setglobal sprsGEMsolve` – set this equal to 1 to invoke `GEMdata.gms` but not `GEMsolve.gms`. The reason to suppress the invocation of `GEMsolve.gms` is to load a newly edited input data set, and generate and inspect the resulting input data summary files without proceeding to solve the model. Otherwise leave `sprsGEMsolve` set to 0.

(k) `$setglobal Mode` – set this to 0 if a GAMS developer license is being used and to 1 if a runtime license is being used.

(l) `$setglobal GRscheduleFile` – specify the location (file path) and name of a file containing a previously determined capacity expansion plan. Such a GAMS-readable ("GR") file is produced every time the model is solved if `GRscheduleWrite` in `GEMsettings.inc` is set to 1. The use of a previously determined capacity expansion plan implies the solve steps `timing` and `reopt` will be skipped in the current run version.

(m) `$setglobal GRscheduleRead` – set this equal to 1 if a previously determined capacity expansion plan file is to be used. Otherwise leave it set equal to 0.

(n) `$setglobal ProgPath` – this variable points to the GEM programs directory, which, if the GEM directory structure complies with the recommended approach, will be the current directory. In other words, it should not require editing.

(o) `$setglobal DataPath` – use this to specify the location of the GEM data directory relative to the GEM programs directory. The standard GEM directory structure means this should not require editing.

(p) `$setglobal OutPath` – use this to specify the location of the GEM output directory relative to the GEM programs directory. The standard GEM directory structure means this should not require editing.

5.7 Note that the values assigned to the `$setglobal` variables should avoid unusual characters and punctuation symbols. As a precaution, it is advisable to enclose the entire string inside a set of double quotes.

Edit GEMsettings.inc

5.8 Using a text editor, open the file called `GEMsettings.inc` in the GEM programs directory and assign values, or character strings, to the following `$setglobal` variables and `scalars`:

(a) `$setglobal firstYear` – specify the first modelled year for the current run version, e.g. 2012. Note that the input data set must contain data for the specified first year.

(b) `$setglobal lastYear` – specify the last modelled year for the current run version, e.g. 2030. The set of modelled years, `y`, in GEM will contain a continuous sequence from `firstYear` up to and including `lastYear`. Obviously the input data set should also contain data for all of the years in set `y`.

(c) `$setglobal RunType` – set equal to 0, 1 or 2 in order to, respectively, indicate whether both `timing`/`reopt` and `dispatch`; just `timing`/`reopt`; or just `dispatch` models are to be solved. The role of the `runType` switch is to determine which variable levels and constraint marginal values should have results collected. This switch could actually be done away with and replaced with an automated detection method.

(d) `$setglobal GEMtype` – specify whether GEM, the model used to solve the `timing` and `reopt` steps, is to be a MIP or an RMIP. Generally speaking, GEM should be solved as a MIP.

(e) `$setglobal DISPtype` – specify whether DISP, the model used to solve the `dispatch` step, is to be an RMIP or an LP. Generally speaking, DISP should be solved as an RMIP, which is essentially an LP. Is DISPtype and GEMtype even still required or can they be removed and/or automated?

(f) `$setglobal calcInputLRMCs` – set equal to 1 in order to have a file of LRMCs by plant created each time `GEMdata.gms` is invoked. Otherwise set it equal to 0.

(g) `Scalar GRscheduleWrite` – set equal to 1 in order to have `GEMsolve.gms` create a GAMS-readable ("GR") file containing a capacity expansion plan for each such plan solved for in each run version. Otherwise set it equal to 0. The reason to create a GAMS-readable capacity expansion plan is to enable subsequent GEM runs or run versions to be given an existing capacity expansion plan.

(h) `Scalar hydroOutputScalar` – specify a factor by which the hydrology data is scaled. Setting it equal to 1 implies no scaling. This scalar only applies to cases where the hydro sequence type is same – is this really what we want to do?

(i) `Scalar WACCg` – specify the WACC to apply to generation investors, e.g. 0.07.

(j) `Scalar WACCt` – specify the WACC to apply to transmission investors, e.g. 0.07.

(k) `Scalar discRateLow` – specify a low discount rate to apply to post-solve sensitivity analysis of the discount rate, e.g. 0.05.

(l) `Scalar discRateMed` – specify a medium discount rate to apply to post-solve sensitivity analysis of the discount rate, e.g. 0.07.

(m) `Scalar discRateHigh` – specify a high discount rate to apply to post-solve sensitivity analysis of the discount rate, e.g. 0.09. Does it make sense to leave this discount rate stuff and other post-solve settings in `GEMsettings.inc`, when solving is now separated from reporting?

(n) `Scalar taxRate` – specify the corporate tax rate, e.g. 0.28.

(o) `Scalar depType` – select a depreciation type; 1 for declining value, 0 for straight line.

(p) `Scalar txPlantLife` – specify the expected life of transmission equipment, e.g. 60 years.

(q) `Scalar txDepRate` – specify the depreciation rate to apply to transmission equipment, e.g. 0.06.

(r) `Scalar randomCapexCostAdjuster` – specify the bounds for a small +/- random adjustment to generation plant capital costs, e.g. 0.05 or 5%. Set equal to 0 if no adjustment is desired. The adjustment only applies to the technologies listed in the set called `randomiseCapex` in the standard input GDX file.

(s) `Scalar txLossesRMIP` – set equal to 1 if the RMIP method is to be employed to compute transmission losses. Otherwise set to 0 to use the MIP method. See constraints `calcTxLossesMIP` and `calcTxLossesRMIP` in `GEMdeclarations.gms` for further details.

(t) `Scalar V2GtechnologyOn` – set equal to 1 to turn on any plant of the `V2G` (vehicle to grid) technology type. If set equal to 0, all V2G plant have their nameplate capacity set equal to zero. It is intended that V2G plant be available to GEM when an energy demand file that accounts for increased demand in the future due to electric vehicles is used.

(u) `Scalar renNrgShrOn` – set equal to 1 to make use of the renewable energy share constraint (`minReq_RenNrg`). Otherwise set equal to 0.

(v) `Scalar renCapShrOn` – set equal to 1 to make use of the renewable capacity share constraint (`minReq_RenCap`). Otherwise set equal to 0.

(w) `Scalar niNWpeakCnstrntOn` – set equal to 1 to make use of the North Island no-wind peak security constraint. Otherwise set equal to 0.

(x) `Scalar limitNrgByFuelOn` – set equal to 1 to employ the constraint that limits energy output by plant fuel type. Otherwise set equal to 0. If the constraint is on, it applies only to fuel types for which the parameter `i_maxNrgByFuel` is non-zero.

(y) `Scalar reservesOn` – set equal to 1 to make use of the detailed reserves formulation in GEM. Otherwise set equal to 0.

(z) `Scalar DCloadFlowOn` – set equal to 1 to turn on the DC load flow formulation. Otherwise leave equal to 0 to formulate and solve GEM as a transhipment problem.

(aa) `Scalar cGenYr` – specify a year, e.g. 2025, from which point all integerized generation build decisions become continuous. The reason for allowing binary decisions to become fractional at some distant future date is to lessen the computational overhead of solving GEM. Specifying a year beyond the last modelled year, `lastYear`, result in no integer decisions becoming continuous.

(bb) `Scalar AnnualMWlimit` – specify an upper bound, e.g. 1000, on the total MW of new generation plant able to be commissioned nationwide in any single year.

(cc) `Scalar noRetire` – specify the number of years, e.g. 2, following and including the first modelled year for which endogenous generation plant retirement decisions are prohibited.

(dd) `Scalar VOLLcap` – specify the nameplate capacity, e.g. 500 MW, of each VOLL or shortage plant that GEM will automatically create in each region, i.e. each region will have a VOLL plant, and all of them will have the same capacity.

(ee) `Scalar VOLLcost` – specify the VOLL cost, e.g. 500/MWh, of each VOLL plant that GEM will automatically create in each region.

(ff) `Scalar penaltyViolatePeakLoad` – specify a $/MWh penalty for failing to meet the peak load constraints.

(gg) `Scalar penaltyViolateRenNrg` – specify a $/MWh penalty for failing to meet the renewable energy constraint.

(hh) `Scalar slackCost` – specify an arbitrarily high cost for slack variables, i.e. constraint violations, to be able to enter the objective function. Unlike penalties that permit the specification of soft constraints, a solution containing slack variables should not be accepted. The slack variables only exist because it is more useful to examine a feasible solution than an infeasible one.

(ii) `Scalar noVOLLblks` – specify the number of contiguous load blocks at the top end of the load duration curve for which VOLL generators are unavailable.

(jj) `$setglobal NumVertices` – specify the number of vertices, or corners, in the piecewise linear interregional transmission loss functions. The number of vertices will be one more than there are segments in the piecewise linear loss function.

(kk) `$setglobal AClossesNI` – specify an upwards adjustment of load on an island-wide basis to account for the intraregional or AC transmission losses in the North Island that are not otherwise modelled.

(ll) `$setglobal AClossesSI` – specify an upwards adjustment of load on an island-wide basis to account for the intraregional or AC transmission losses in the South Island that are not otherwise modelled.

(mm) `$setglobal Solver` – specify which LP/RMIP/MIP solver to use. Options may include Cplex, Gurobi, or Xpress.

(nn) `$setglobal optCr` – specify the value for the GAMS relative optimality criterion, optcr, used to solve MIPs.

(oo) `$setglobal CPUsecsGEM` – specify the CPU seconds available to the solver when solving the GEM model. It is important to give MIPs a time limit in case a model instance has been configured that is unable to ever reach the specified optimality criterion.

(pp) `$setglobal Threads` – specify the number of threads the available MIP solver is licensed to use.

(qq) `$setglobal limitOutput` – set to 1 to restrict the amount of output written to the GEMsolve listing file. Otherwise set it to 0. If many models are solved in one run version, a very large listing file will be created that becomes useless because it is too big to be opened.

5.9 Most of the settings in `GEMSettings.inc` will not require changing from one run or run version to the next.

Edit GEMstochastic.inc

5.10 `GEMstochastic.inc` is quite complex and may be difficult for a novice user to edit using a text editor. It is intended that the EMI be used to produce `GEMstochastic.inc`. However, in the absence of EMI, the graphical utility application called `createGEMstochastic` can and should be used. This utility is distributed with GEM and can be found in the GEM programs directory – double-click it to invoke it.

5.11 The editing of `GEMstochastic.gms` is necessary to:

(a) create the scenarios, scenario sets, and experiments;

(b) map scenarios to scenario sets, historical hydro years, and hydro sequence types;

(c) map scenario sets to experiments;

(d) assign values to scenario-specific factors;

(e) assign weights to the scenarios mapped to each scenario set; and

(f) select one scenario to be the default scenario.

5.12 The purpose of the default scenario is to restrict the amount of information reported in the input data summary files. All input data associated with the specified default scenario is summarised whereas only the manner in which non-default scenarios differ from the default is summarised. The choice of default scenario is somewhat arbitrary. It simply reduces repetition in a file purporting to be a summary.

Invoke runGEMsetup.gms

5.13 Once the input data set has been created, and the three aforementioned .inc files (`GEMpathsAndFiles`, `GEMsettings`, and `GEMstochastic`) have been appropriately edited, the user is now in a position to run or solve GEM. This task is undertaken by invoking a series of so-called `runGEM` scripts or programs. The first of these is called `runGEMsetup.gms`.

5.14 `runGEMsetup.gms` should be invoked just the once to set up a GEM run.

5.15 After importing `GEMpathsAndFiles.inc`, the `runGEMsetup.gms` script sets up a location in the GEM output directory where the output of the GEM run will be placed. It also archives certain

files so that the precise configuration of the GEM run can be examined and the job replicated, if required, at some future date. Depending on the type of GAMS license, GEMdeclarations.gms may automatically be invoked. Non-developer users of GEM should have no need to edit runGEMsetup.gms or GEMdeclarations.gms.

Invoke runGEMDataAndSolve.gms

5.16 runGEMdataAndSolve.gms should be invoked once for every run version of the current run. Between each invocation, the user would presumably edit GEMpathsAndFiles.inc so as to point to an alternative input GDX file or override file, or modify some values in GEMsettings.inc. One way or another, something needs to be different about each run version otherwise, obviously, the solution to each run version would be identical.

5.17 After importing the file called GEMpathsAndFiles.inc, this script sequentially invokes the two programs called GEMdata.gms and GEMsolve.gms. GEMdata.gms itself will import GEMpathsAndFiles.inc, GEMsettings.inc, and GEMstochastic.inc. The GEM code called GEMdata.gms then proceeds to load the data from the three input GDX files specified by the user in GEMpathsAndFiles.inc and performs various computations to get the input data ready to pass along to GEMsolve.gms.

5.18 GEMsolve.gms actually solves the model, or models, as configured by the user in GEMstochastic.inc. In addition, GEMsolve.gms will create GDX files of the model solution (all variable levels and equation marginal values). These GDX files can be examined directly by users, or, more likely, will be fed into the GEM reporting process.

5.19 Non-developer users of GEM should have no need to edit runGEMDataAndSolve.gms, GEMdata.gms or GEMsolve.gms.

5.20 It should be apparent by now that a run version may involve solving many model instances. For a variety of reasons, most having to do with inappropriately selected input data values, one or more model instances may terminate without finding an optimal solution. Before producing reports, users should be certain that all model instances in each run and run version for which reports are to be generated have been successfully solved. In the root of the output directory for the current run, there will be a file called GEMsolveReport – runName.txt. This file should be inspected to ensure that each model instance of each run version in the current run has been solved successfully. For example, does the objective function value look sensible? Are slack variables present in any or all model solutions? Do the experiments-steps-scenarioSet tuples appear in the order and combinations expected?

5.21 In the case where one or more model instances have not been solved successfully, users should investigate why, take corrective action, and re-solve the offending run versions before proceeding to generate reports.

Edit GEMreportSettings.inc and runGEMreports.gms

5.22 As previously noted, the process of generating reports is distinct from solving the model(s). Consequently, it is necessary to instruct the report writer as to what previously solved model runs and run versions are to be included in the reporting process.

5.23 The easiest way to generate reports is to invoke the utility called createRunGEMreports, which is distributed with GEM and should be found in the GEM programs directory. Invoking this utility and making the appropriate selections will then cause two files to be produced: GEMreportSettings.inc and runGEMreports.gms. Subsequently invoking runGEMreports.gms will then cause GEMreports.gms to read in the specified GDX files and

produce a collection of reports. The reports will be located in an output directory whose name begins with the suffix `rep` and continues with the user-supplied report name.

5.24 Alternatively, `GEMreportSettings.inc` and `runGEMreports.gms` can be manually edited using a text editor. This task may be quite complicated for the novice user.

5.25 Up to here with editing of this doc….

5.26 The contents of `GEMreportSettings.inc` are as follows:

(a) `$setglobal reportName` – specify a name for the report. This name will be used to create the directory where the reports will be located

(b) `$setglobal BaseCaseRun` – nominate one of the runs to be reported on to be the base case run. There may only be one run to be reported on.

(c) $setglobal BaseCaseRV – nominate one of the run versions in the base case run directory to be the base case run version. The purpose of the base case run and run version is to identify an input set of data from which the membership of fundamental sets are drawn, e.g. set `y`, the modelled years; set `g`, the generating plant, set `k`, the technologies; etc. It is up to the user to ensure that the run versions to be reported on for any given report all use the same fundamental sets. It makes little sense to simultaneously report on two model solutions using different sets of generating plant, for example.

(d) `Set runVersions` – this set must specify all run versions currently being reported on.

(e) `Set runVersionColor(runVersions,*,*,*)` – the RGB color mix for the run versions being reported on.

(f) `$setglobal ProgPath` – this variable points to the GEM programs directory, which, if the GEM directory structure complies with the recommended approach, will be the current directory. In other words, it should not require editing.

(g) `$setglobal DataPath` – use this to specify the location of the GEM data directory relative to the GEM programs directory. The standard GEM directory structure means this should not require editing.

(h) `$setglobal OutPath` – use this to specify the location of the GEM output directory relative to the GEM programs directory. The standard GEM directory structure means this should not require editing.

5.27 The contents of runGEMreports is….

Invoke runGEMreports.gms

5.28 After importing the file called `GEMreportSettings.inc`, this script takes the GDX files produced by GEMsolve.gms and merges them into a single GDX file ready for use by `GEMreports.gms`. More specifically, a set of merged GDX files is required for each variant of the run that is to be reported on. After merging the relevant GDX files, `runGEMreports.gms` will automatically invoke `GEMreports.gms`, which in turn will do all the work to create the reports.

Inspect GEM output

5.29 Go to the GEM output directory called `repReportName` and go wild. There you will find more GEM results than you can poke a stick at.

5.30 XXX

Appendix A [text]

Glossary of abbreviations and terms

.gms	A text file containing GAMS code
.inc	A text file to be included in a .gms file using the $include command
.lst	A GAMS listing file. Each time a .gms file is invoked, a .lst file of the same name is created. The .lst file can be inspected when things go wrong.
Authority	Electricity Authority
CSV	Comma separated variable - Microsoft jargon for a good old fashioned comma-delimited text file. GEM CSV files will enclose text strings inside double-quotes.
EMI	Electricity market information - a graphical user interface for selected Electricity Authority models
GAMS	General algebraic modeling system - a software application used to solve optimisation models
GEM	Generation expansion model
IDE	Integrated development environment
LRMC	Long-run marginal cost
MW	Megawatt
MWh	Megawatt hour
SRMC	Short-run marginal cost
VOLL	Value of lost load
WACC	Weighted average cost of capital