GridDyn Users Guide
Version 0.5
Friday 12th August, 2016

Philip Top Ph.D.

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.
LLNL-SM-700378

Contents

1 Introduction

GridDyn is a power system simulator developed at Lawrence Livermore National Laboratory. The name is a concatenation of Grid Dynamics, and as such usually pronounced as "Grid Dine". It was created to meet a research need for exploring coupling between transmission, distribution, and communications system simulations. While good open source tools existed on the distribution side, the open source tools on the transmission side were limited in usability either in the language or platform or simulation capability, and commercial tools, while quite capable, simply did not allow the access to the internal components and data required to conduct the research. Thus, the decision was made to design a platform that met

the needs of the research project. Building off of prior efforts in grid simulation, GridDyn was designed to meet the current and future research needs of the various power grid related research and computational efforts. It is written in C++, making use of recent improvements in the C++ standards. It is intended to be cross platform with regard to operating system and machine scale. The design goals were for the software to be easy to couple with other simulation, and be easy to modify and extend. It is very much still in development and as such, the interfaces and code is likely to change, in some cases significantly as more experience and testing is done. It is our expectation that the performance, reliability, capabilities, and flexibility will continue to improve as projects making use of the code continue and new ones develop. We expect there are still many issues so any bug reports or fixes are welcome. And hopefully even in its current state and as the software improves the broader power systems research community will find it useful.

2 Installation

GridDyn is written in C++ and makes use of a few external libraries not included in the released source code. External software packages needing installation prior to compilation of GridDyn include

- Boost 1.61 –for most parts of GridDyn Boost 1.58 or greater will be sufficient but going forward some part of GridDyn will be making use of libraries only found in 1.61 or higher. GridDyn currently compiles with Boost version >1.49.
- SUNDIALS 2.6.2 (GridDyn will be updated to use the new version of SUNDIALS when it is released later in 2016 at which time that will become the required version)
- KLU in the suitesparse package.
- cmake for building
- Doxygen for building in source documentation.
- all other third party code is included in the release.

Boost can be downloaded from `www.boost.org` many of the features in GridDyn will work with older versions but going forward we will be making use of some features from Boost 1.61 and will use that as the baseline going forward. SUNDIALS can downloaded at `http://computation.llnl.gov/sundials` the current version is 2.6.2 but will be upgraded to the new version when it is released in the near future. SUNDIALS should be built with KLU support enabled for reasonable performance. KLU is part of SuiteSparse on most Linux type systems it can be installed as a package, on Windows cmake files can be found at `https://github.com/jlblancoc/suitesparse-metis-for-windows`.

GridDyn uses a cmake build system to construct build files for whatever platform you happen to be on (assuming it is supported by cmake) GridDyn uses C++11 extensively and will make use of some C++14 features in the near future. Therefore, required future compilers are

- Visual Studio 2015
- gcc 4.9.3 or higher (4.8 works for the moment but will not in near future updates)
- clang 3.5 or higher (openMP must be turned off to use 3.4)
- Intel 16.0 (not thoroughly tested as of yet)

At present GridDyn will likely compile on gcc 4.8 and visual studio 2013 but that is expected to change with future updates.

2.1 Cmake options

BOOST_ROOT if boost is not found in the system directories or a different version is desired the root location of boost can be specified in BOOST_ROOT

BUILD_SERVERMODE the servermode for GridDyn is a work in progress, it is recommended that this option be disabled pending further progress

DEBUG_LOG_ENABLE unselecting disables all DEBUG and TRACE log messages from getting compiled

TRACE_LOG_ENABLE unselecting disables all TRACE log messages from getting compiled

DOXYGEN_OUTPUT_DIR location for the generated doxygen documentation

ENABLE_64BIT_INDEXING select if you want support inside GridDyn for more than $2^{32} - 2$ states or objects (I find this unlikely at this time)

ENABLE_EXPERIMENTAL_TEST_CASES select to enable some experimental test cases in the test suite

FMI_ENABLE enable experimental support for FMI objects. This is in development, recommended to leave unselected unless you are developing

FSKIT_ENABLE enable to build additional libraries and support for integration into FSKIT and PARGRID for tool coupling

GRIDDYN_GENERATE_DOXYGEN_DOC select to create a case to build the doxygen docs

GRIDDYN_OPENMP (doesn't do any thing yet) eventually it will enable openMP in the GridDyn evaluation functions

KLU_ENABLE this option may be removed in the future recommended to leave selected otherwise KLU support will not be built in

KLU_INSTALL_DIR point to the installation directory for KLU if it was not found in the system directories

LOAD_ARKODE build in support for ARKODE for solving differential equations

LOAD_CVODE build in support for CVODE for solving differential equations Neither cvode or arkode are used at present but will be in the near future

LOAD_EXTRA_MODELS select to build an additional library containing a few optional models-more will likely be added in the future

LOG_ENABLE unselect to turn off all logging functions

MPI_ENABLE select to build with MPI support using an MPI compatible compiler

OPENMP_ENABLE option to build in support for openMP in both the solvers and in GridDyn

OPTIMIZATION_ENABLE enable building of the optimization extension. This is a work in progress and doesn't do much yet, recommended to leave unselected unless you are developing on that section.

SUNDIALS_INSTALLATION_DIR point to the installation location for SUNDIALS

SUNDIALS_OPENMP select to enable OpenMP in SUNDIALS (assumes SUNDIALS was built with openmp support)

TEST_ENABLE enable building of the test-suites

THREAD_ENABLE not used at preset but will eventually enable threaded execution in some models

2.2 Installation Notes

2.2.1 Mac

Example of successful build on a mac OS X GridDyn on Mac OS X

1. Install MacPorts
2. Install cmake port
3. Download SuiteSparse 4.2.1 `http://faculty.cse.tamu.edu/davis/suitesparse.html`
4. Build/install SuiteSparse
5. Download SUNDIALS 2.6.2 `http://computation.llnl.gov/projects/SUNDIALS-suite-nonlinear-differential-algebraic-equation-solvers/download/SUNDIALS-2.6.2.tar.gz-` didn't build dynamic libs because they caused issue with GridDyn runtime
6. Configure/build/install SUNDIALS (cd SUNDIALS-2.6.2; mkdir build; ccmake ..; update paths for your install location (prefix, klu lib and include dirs), set KLU_ENABLE to ON); make; make install)

7. Download/configure/build/install BOOST

8. Configure GridDyn (cd transmission; mkdir build; cd build; ccmake -DKLU_DIR=< ... > -DBoost_INCLUDE_DIR=< ... > -DBoost_LIBRARY_DIRS=< ... >;) or use cmake-gui

2.2.2 Linux

Depending on the distribution, Boost or an updated version of it may need to be installed. SUNDIALS and KLU may need to be installed as well. Typically camke is used to generate makefiles thought it has been used to generate Eclipse projects. BOOST_ROOT, SUNDIALS_INSTALLATION_DIR, and KLU_INSTALL_DIR may need to be user specified if they are not in the system directories. This can be done in the cmake-gui or through the command line tools. Them running make will compile the program. Running make install will copy the executables and libraries to the install directory.

2.2.3 Windows

GridDyn has been successfully built with Visual Studio 2015 and on Msys2. The msys2 build is much like building on Linux This works fine with gcc, the current clang version on msys2 has library incompatibilities with some of the boost libraries due to changes in gcc. I don't fully follow what the exact issue is on but clang won't work on Msys2 to compile GridDyn unless SUNDIALS, boost, and KLU are compiled with the same compiler, I suspect the same issue is also present in some other Linux platforms that use gcc 5.0 or greater as the default compiler. The suitesparse version available through pacman on msys2 seems to work fine.

For compilation with Visual Studio boost will need to built with the same version as is used to compile GridDyn. Otherwise follow the same instructions.

2.3 Included Third party Libraries

A few small libraries are included in with the GridDyn Source code. The primary XML reader is the tinyXML library along with the the tinyXML++ wrapper. TinyXML2 is also used in a few circumstances. Both were written by Lee Thomason. Both are released under a Zlib license. TinyXML++ is a wrapper around tinyxml making use of C++ constructs written by Ryan Pusztai and Ryan Mulder released under the same license as TinyXML. The code was modified slightly in a few cases to remove deprecated instances of auto_ptr and replace with unique_ptr. The included Json reader is from the JsonCpp project written by Aaron Jacobs and is licensed under an MIT license. GridDyn uses an amalgamated source constructed from the library.

FMI functionality is provided by the FMI library 2.0 released under BSD license as part of the JModelica package.

3 Design Philosophy

GridDyn was formulated as a tool to aid in research in simulator coupling. Its use has expanded but it is primarily a research tool into grid Simulation and power grid related numeric methods, and designed and constructed to enable that research. It is open source, released under a BSD license. All included code will have a similarly permissive license. Any connections with software of other license will require separate download and installation. It is intended to be fully cross platform, enabling use on all major operating systems, all libraries used internally must support the same platforms. However interaction with other simulators, such as for distribution or communication may impose additional platform restrictions. Optional components may not always abide within the same restrictions.

Prior to release 1.0 very little effort will be expended in backwards compatibility. GridDyn was written making extensive use of C++11 constructs, and will shortly be making more use of C++14 standard constructs. Specifically allowing any features of the standard supported by GCC 4.9.X versions. It is expected this will be the minimum version until newer compilers are much more widely accessible.

3.1 Modularity

GridDyn code makes heavy use of object oriented design and polymorphism and is intended to be modular and replaceable. The design intention is to allow users to define a new object that meets a given component specification and have that be loaded into the simulation as easy as any previously existing object, and require no knowledge of the implementation details of any other object in the simulation. Thus allowing new and more complex models to be added to the system with no disruption on the rest of the system. Models also do not assume the presence of any other object in the system though are allowed to check for the existence first. This is exemplified in the interaction of generators with its subcomponents. Any combination of Generator Model, exciter, and governor should form a valid simulation even though some combinations may not make much physical sense or be realistic.

3.2 Mathematics

The GridDyn code itself has only limited facilities for numeric solutions to the Differential algebraic equations which define a dynamic power system simulation. Instead it relies on external libraries interacting through a solver Interface tailored for each individual solver. The models are intended to be very flexible in support for an assortment of numeric approximations and solution models, and define the equations necessary for model evaluation.

Initial development of dynamic simulation capability is done through a coupled differential-algebraic solver with variable time stepping/ The Primary solver used is IDA from the SUNDIALS package[1]. It can use the dense solver or the KLU sparse solver which is much faster for obvious reasons. Recent work incorporates the use of a fixed time step solution mode, with a partitioned set of solvers separating the algebraic from the differential components and solving them in alternating fashion. At present this is much less well tested. Initial formulations use CVODE for the differential equations and KINSOL for the algebraic solution. Kinsol is also used to solve the power flow solution. ARKODE, an ode solver use solver

using Runga-Kutta methods is also available for solving the ODE portion of the partitioned solution.

In order to provide support for current and future models of grid components a decision was made to distribute the grid connectivity information and not use a Y-bus matrix as is typical in power system simulation tools. This allows loads and transmission lines to be modeled using arbitrary equations. This decision alters the typical equations used to define a power flow solution at buses. each bus simply sums the real and reactive power produced or consumed by all connected loads, generators, and links. Those components are free to define the power an arbitrary function of bus voltage, angle, and frequency, provided that function is at least piecewise continuous. NOTE: Continuous functions work much better, piecewise continuous functions work but don't really play nicely with the variable timestepping. Defining the problem in this way comes at a cost of complexity in the complementation and likely a performance hit but allows tremendous flexibility for incorporating novel loads, generators, and other components into power flow and dynamic simulation solutions. The dependency information is extracted through the Jacobian function call. Currently the solution always assumes the problem is non-linear even if the approximations used are in fact linear. While GridDyn's interaction with the solvers comes exclusively through interface objects, there may be some inherent biases in the interface definition due to primary testing with the SUNDIALS package. These will likely be exposed when GridDyn is tested with alternative numerical solvers.

3.3 Model Definition

GridDyn is intended to be flexible in its model definition allowing details to be defined through a number of common power system formats. The most flexible definition is through a GridDyn specific XML format. Strictly speaking the most flexible XML cannot be defined by an XML schema due to the fact that the readers allow element names to describe properties of which the complete set of which cannot be described due to support for externally defined models. Alternate formulations exist which could be standardized in a schema but not attempt has been made to do so. The XML formulation includes a variety of programming like concepts to allow construction of complex models quickly, including arrays and conditionals, as well as limited support for equations and variable definitions. The file ingest library also supports importing other files through the XML and defining a library of objects that can be referenced and copied elsewhere. The typical use case is expected to be importing a file of another format that contains a majority of the desired simulation information and only defining the solver information and any GridDyn specific models and adjustments in the XML. The general idea is to be as flexible and easy to use as possible for a text based input format, and as GridDyn Develops support as many other formats as is practically possible. All the file ingest functionality is contained in an separate library from the model bookkeeping and model evaluation functionality. Other types of input can be added as necessary and Some development is taking place towards an GUI which would interact through REST service commands and JSON objects. Included in GridDyn are capabilities of searching through objects by name, index number or userID.

3.4 Performance

GridDyn was designed for use in an HPC environment. What that means right now is that GridDyn can interoperate with other simulators in that environment and some considerations were put in place in the design but GridDyn on its own does not really take advantage of parallel processing. As of release 0.5 the transmission power flow and dynamic solve is not itself parallel in any way. Considerable thought has been put into how that might be accomplished in later versions but it is not presently in place. Initial steps will include adding in optional openMP pragmas to take advantage of the inherent independence of the objects in calculation of the mathematical operations such as residual or Jacobian. OpenMP vector operations can be enabled in SUNDIALS though this is only expected to result in small performance gains and only for models over 5000 buses. Further tests will be done to determine exact performance gains.

Some effort has gone into improving the performance of the power flow solve and only incremental gains are expected at this point using the current solve methodology. No effort has been expended on the dynamic simulation so some performance improvements can be expected in that area when examined.

The system has no inherent size limitations. Limited only by memory on any given system. Scalability studies have been carried out to solving a million bus model, It could probably go higher but the practical value of such a single solve is unclear as of yet.

3.5 Model Libraries

The aim thus far in GridDyn has been the development of the interfaces. The models available are the result of programmatic needs or the need to ensure the simulator is capable of dealing with specific kinds of model interactions. As a result the models presently available represent only a small subset of those defined in power system libraries. More will be available as time goes on, but the idea is not to have a large collection internally but to enable testing of new models, and to incorporate model definition libraries through use of other tools and interfaces such as FMI, and possibly others as needed.

3.6 Testing

a suite of test cases is available and will continue to grow as more components and systems are thoroughly tested. The nature of the test suite is evolving along with the code and will continue to do so. It makes use of the BOOST test suite of tools and if built creates 3 executable test programs that test the various aspects of the system. While we are still a ways from that target 100% test coverage is a goal though likely not realistic in the near future. The code is regularly compiled on at least 5 different compilers and multiple operating systems and strives for warning free operation.

3.7 Test Programs

If enabled 3 test programs are built. These programs execute the unit test suite for testing GridDyn. They are divided into 3 programs. testLibrary runs tests aimed at testing oper-

ation of the various libraries used In GridDyn. The testComponents program executes test cases targeted at the individual model components of GridDyn. The third, testSystem, runs system level tests and some performance and validation tests on GridDyn. After installation these test programs are placed in the install directory and can be executed by simply running the executable. Specific tests can be executed with command line parameters

```
> ./testComponents --run_test=block_tests
> ./testComponents --run_test=block_tests//block_alg_diff_jac_test

> ./testLibrary -h
```

4 Executable operation

The main executable for GridDyn is built as gridDynMain and is intended to load and run a single simulation. The executables testSystem, testComponents, and testLibrary are test programs for the unit Testing of GridDyn. A servermode for interactive sessions is a work in progress but is not operational at the time of this release. The primary executable is built as gridDynMain.

```
>./gridDynMain --version
```

will display the version information

```
> ./gridDynMain -h
```

will display available command line options.
typical usage is

```
> ./gridDynMain [options] inputFile [options]
```

the primary input file can be specified with the flag –input or a single flagless argument. additional input files should be specified using -i or –import flags. command Line only options

--help produce help message

--config-file arg specify a config file to use

--config-file-output arg file to store current config options

--mpicount setup for an MPI run, prints out a string listing the number of MPI tasks that are required te execute the specfied scenario, then halts execution.

--version print version string

configuration options

-o, --powerflow-output filename file output for the powerflow solution. extension specifies a type (.csv, .xml, .dat, .bin, .txt) unrecognized extensions default to the same format as .txt.

-P, --param arg override simulation file parameters -param ParamName=

`<val>`

-D, --dir directory add search directory for input files

-i, --import filename add import files loaded after the main input file

--powerflow-only set the solver to stop after the power flow solution

--state-output filename file for saving states, corresponds to –save-state-period

--save-state-period arg save state every N ms, -1 for saving only at the end

--log-file filename log file output

-q, --quiet] set verbosity to zero and printing to none

--jac-output arg powerflow Jacobian file output

-v, --verbose arg specify verbosity output 3=verbose, 2=normal, 1=summary, 0=none

-f, --flags arg specify flags to feed to the solver eg. --flags=flag1,flag2,flag3 no spaces between flags if multiple flags are specified or enclose in quotes

-w, --warn arg specify warning level output for input file processing 2=all, 1=important, 0=none

--auto-capture filename automatically capture a set of parameters from a dynamic simulation to the specified file format is determined by extension. either .csv or .txt will record the output in csv format, all others will record in the binary file format. The filename must be specified with --auto-capture-period if used.

--auto-capture-period arg specifies the automatic capture period in seconds. If specified without a corresponding --auto-capture file. a file named auto_capture.bin is created.

--auto-capture-field arg , specify the fields to be captured through the auto capture, Defaults to "auto". can be a comma or semicolon separated list, no spaces unless enclosed in quotes.

The configuration routine will look for and load a file named gridDynConfig.ini if it is available. It will also load any command line specified config file. The order of precedence is command line, user specified config file, then system config file(if available).

5 Components

Components in GridDyn are divided into three categories: primary, secondary, and sub-Model. Primary Components include buses, links, relays, and areas and define the basic building blocka for power grid simulation. Secondary components are those which tie into Busses and consume or produce real and reactive power. The two component types in the secondary category are loads and generators. Submodels are any other component in the system and can form the building blocks of other components. A majority of the differential equations in the dynamic simulations are found in submodels. Submodels include things such as exciters, governors, generator models, control systems, sources, and several others. There are a few other types of objects used in GridDyn but they generally are used for specific purposes and do not take part in the equations unless interfaced through another object. The component types currently available in GridDyn will be detailed later in this document.

5.1 Buses

Buses form the nodes of a power system. They act as containers for secondary objects and attach to links. The default bus type is an ac bus which in typical operation would have 2 states (voltage and angle). 4 types of bus operation are available PQ, PV, slack and fixed Angle. The practical value of fixed angle buses is unknown but was included for mathematical completeness and describes a bus whose angle and reactive power are known. The residual equation used in the bus model take one of two forms

$$f_v(X) = \sum_{i=0}^{gens} Qgen_i(V,\theta,f) + \sum_{i=0}^{loads} Qload_i(V,\theta,f) + \sum_{i=0}^{linest} Qline_i(V,\theta,f) \tag{1}$$

for PQ and afix type buses and

$$f_v(X) = V - V_{target} \tag{2}$$

for PV and SLK type buses. The equations for θ are very similar

$$f_\theta(X) = \sum_{i=0}^{gens} Pgen_i(V,\theta,f) + \sum_{i=0}^{loads} Pload_i(V,\theta,f) + \sum_{i=0}^{lines} Pline_i(V,\theta,f) \tag{3}$$

for PQ and PV type buses and

$$f_v(X) = \theta - \theta_{target} \tag{4}$$

for fixed angle and SLK type buses.

the frequency can be either extracted from an active generator attached to the bus or computed as a filtered derivative of the angle. If it is computed the bus has an additional state as part of the dynamic calculations.

The Bus Model implementation in GridDyn also includes some ability to merge buses together to operate in node-breaker type configurations. At present this is not well tested.

5.2 Areas

Areas define regions on the simulated grid. An area can contain other areas, buses, links, and relays. It principally acts as a container for the other objects, though will eventually include controls such as AGC and other wide area controls. The simulation object itself is a specialization of an area.

5.3 Links

In the most general form links connect buses together. As a primary object it can contain other objects, include state information. The basic formulation is that of a standard AC transmission line model connecting two buses together. The code includes a number of possible approximations

5.4 Relays

Relays are perhaps the most interesting and unusual primary object included in GridDyn. The basic concept is that relays can take in information from one abject and act upon another. They add protection and control systems into the simulation environment. They exist as primary objects since they can stand on operate on their own at the same level as buses and areas. They may contain states, other objects, submodels, etc. They also act as gateways into communication simulations, functioning as measurement units and control relays. And through relays a whole host of control and protection schemes can be implemented in simulation alongside normal power flow and dynamic simulations. Examples of relays include fuses, breakers, differential relays, distance relays, and control relays among others.

6 Development Notes

GridDyn is very much a work in progress, development is proceeding on a number of different aspects from a number of directions and many components are in states of partial operation or are awaiting development in other aspects of the code base. The notes in this section attempt to capture the development status of various GridDyn components and note where active and planned development is taking place.

6.1 interface and executables

A gridDynServer executable is in development. This program will become the main means of interacting with simulations. The plan will be gor it to support multiple running simulations and allow users to interact through a set of interfaces. Planned interfaces include A RESTFUL services inteface for ethernet based interaction, which will eventually be the basis of interaction with a GUI, a command line interface, and a direct application interface through TCP/UDP or MPI.

Also in development is a wrapper around the simulation engine into a functional Mockup interface to allow GridDyn to interact with other simulations through the FMI for co-simulation framework.

6.2 Models

The Models included in GridDyn are an evolving set. They have been added to address particular research questions or needs or test specific aspects aspects of GridDyn operation. THe next several subsections talk about the state of development in the various components available in GridDyn

6.2.1 Buses

The bus code is well tested but is constantly evolving to simplify the code or areas of responsibility, or to improve operation. Even though the equations used in the bus evaluation are quite straightforward. The bus itself is one of the more complex objects in gridDyn in order to handle the management of loads and generators and the associated limits and controls. As well as the associated transition between powerflow and dynamic simulation. Currently available are an ACbus, a DC bus for association with HVDC transission lines a trivial bus, and an infinite bus. Some plans are in place for a 3-phase bus but that has been low on the priority list. The DC bus is not thoroughly tested, particularly in dynamic contexts.

6.2.2 Area

At present areas are primary used as a way to group objects. Ongoing development is taking place to add in area wide controls such as AGC. Some of these structures are in place but have yet to be tied in with the Area model itself. There is work ongoing to do this and some form will be functional within the next 3 months. Areas and subareas can be configured through the GridDyn XML format but none of the other available formats such as CDF or PTI currently make use of the area information available in those formats. This will be added alongside the development of area controls.

6.2.3 Links

The basic AC link has been tested thoroughly in powerflow and dynamic simulations by comparison with standard test cases. Other link models such as DC links, and an adjustableTransformer model have been tested in power flow simulations, but the dynamics of them are a work in progress. They operate fine in that context but do not include the control dynamics at least at a level that is well-tested.

6.2.4 Relays

The generic relay is one of the more complex objects to setup. Most use cases involve using one the specific relay types as they embody the information for setting up a relay. There are no known issues with the relays though given their complexity it is likely there are many

circumstances when they do not function appropriately, or a cause issues with interaction of the other parts of the system. The basic relay contains tremendous flexibility and it is not recommended that beginning users attempt to directly instantiate it. You are of course welcome to try but the specification of conditions and actions is somewhat more complex than most other system properties through the XML. Other relay types are in development as needed by specific usage requirements.

6.2.5 Loads

A number of types of loads are modeled in GridDyn. The basic Model is a ZIP model. Extensions include ramps and a variety of other load shapes and others such as an exponential load and frequency dependent load. Also included are motor loads, including models of first order, 3rd order and 5th order induction models, and include mechanisms for modeling motor stalling. The 5th order model has some potential issues during certain conditions that have not been fully debugged. All work in powerflow and dynamic simulation. Code for loading a GridlabD distribution system is included in the release but will not function without corresponding alterations to a GridlabD instance and operation with the Pargrid, neither of which are included in this release, so for all practical purposes it will revert to a debug mode with a simulated distribution simulation intended for debugging operations. The actual functionality necessary for coupling with a distribution system will hopefully be released in the near future, thought could be made available for partners.

There is a composite load model available. This is a more generic container for containing other load models. This is distinct and more general than the composite load model defined by FERC. Though an instantiation of that model is planned and will make use of the generic composite model in GridDyn.

6.2.6 Generators

Two basic generators are available a regular generator and a variable generator intended as a base for modeling renewable generation. The Generator model itself contains very little in the way fo mathematical functions instead it acts a manager for the various submodels that make up a power system generator. These include governors, exciters, generator models, and Power system stabilizers. The variable generator also has mechanisms for including sources which are data generators, and filters. The combination of which creates a mechanism for feeding weather data to a solar or wind plant and converting that into power. The generator is specifically formulated to allow any/all/or none of the subcomponents to be present and still operate. A default generator model is put input place if none is specified and a dynamic simulation is required. A third generator which includes a notion of energy storage is in planning stages.

6.2.7 Generator Models

A wide assortment of genModels is included. Most have been debugged and tested. The classical generator model includes a notion of a stabilizer due to inherent instabilities under fault conditions when attached to an exciter and or governor. Not that classical generator model is an appropriate model to use for such circumstances but nonetheless a stabilizer was

incorporated to make the model stable. The incorporation of saturation into the models is not complete. The models accept the parameters but are not included in the calculation. GENROU and GENSAL models are being developed but are not complete as of release 0.5.

6.2.8 Exciters

available exciters include simple, IEEE type1, IEEE type2, DC1A, and DC2A. The DC2A model has some undiagnosed issue in particular situations and is not recommended for use at present.

6.2.9 Governors

The basic governor and TGOV1 models are operational others are not completed and further work is being delayed until a more general control system model is in place which will greatly simplify governor construction as well as other control systems. The deadband is not working in TGOV1.

6.2.10 Power System Stabilizers

The current Pss code is a placeholder for future work. No PSS model is currently available, though some initial design work has taken place. The work has been delayed until the controlSystem code is operational.

6.2.11 Control Blocks

control blocks are a building block for other models and a number of them are used in other models throughout GridDyn. Development on the generic transfer function block is not finished but the others are working and tested. These will form the building blocks of a set of general control system modules which could be used to build other types of more complex models.

6.3 Others

Other components in GridDyn include Sources which are operational but not well tested in practice, schedulers which are used to control generator scheduling, and other types of controllers for AGC, dispatch, and other sorts of controls. Most of these are various states of development and not well tested.

6.3.1 Events

GridDyn Supports a notion of events which can be scheduled in a simulation and can basically alter any property of the system with the exception of some models prohibiting changing of certain properties after simulation has begun, in this case the event will still be valid, just won't do anything. Support for more complex events involving multiple devices in a more straightforward fashion is planned.

6.3.2 Recorders

Support for extracting any calculated field or property from any object is supported through grabber objects. This can be done directly via the state arrays or from the objects themselves. The files can be saved periodically or at the end of the simulation in a binary format or in CSV. Readers for the binary format are available in C++, matlab, and Python. If a large amount of data is captured frequently for dynamic simulations there is a currently a performance hit. There are ideas for mitigating this that will be addressed when the performance of the dynamic simulation is studied and addressed.

6.3.3 Simulation

Some of the mechanics and interfacing of the planned optimization extension are in place but nothing actually works yet, so don't use it.

6.3.4 FMU Interaction

This works in some cases but is a little more complex to set up than the rest of the code as it is under significant active development, therefore it is not recommended for use at this time.

6.4 File Input

GridDyn is capable of reading XML and Json files defining the GridDyn data directly these formats can take advantage of all GridDyn Capabilities. Json is not as well tested and was targeted mainly for the server interface, but it should work as a file format just fine. A fairly flexible CSV input file reader is also available for inputing larger data sets in a more workable format. CDF files are read though the area and a few other properties not important for powerflow are a not loaded into GridDyn yet. Most of the common elements in raw and pti files are also loaded properly. Some of the more exotic elements such as multiterminal DC lines and 3 way transformers are not yet, mainly since we have no examples of such things in example files. EPC files for PSLF are the same though used less extensively than raw files. Matlab files from Matpower and PSAT can also be loaded. Not all dynamic models from PSAT are available, For DYR files models that match those available are loaded and some others are translated to available models. The library of models in GridDyn is much smaller than those available in commercial tools. Support for other formats is added as needed by projects.

7 XML Input

The following section contains a description of the XML input file format and how to construct and specify an input file in the GridDyn XML format. As stated previously the XML format is intended to be used solely in GridDyn to enable full access to all the capabilities and models that may or may not be defined in other formats. All the actual interpreters have been designed to use an element tree structure. And as such the same reader code is

used for the XML interpreter and for a JSON interpreter, though there is some variance in the definitions of elements and attributes in those two contexts meaning Json objects are somewhat more restricted in format. In the documentation a most of the examples will be in XML, but a few will be in JSON for completeness.

7.1 Initial Example

A simple input case is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<GridDyn name="2bus_test" version="1">

<bus name="bus1">
    <type>SLK</type>
    <angle>0</angle>
    <voltage>1.05</voltage>
    <generator name="gen1">
    </generator>
    <load name="load1">
        <P>1.05</P>
        <Q>0.31</Q>
    </load>
</bus>

<bus name="bus2">
    <load name="load2">
        <P>0.45</P>
        <Q>0.2</Q>
    </load>
</bus>
<link from="bus1" name="bus1_to_bus2" to="bus2">
    <b>0.127</b>
    <r>0.0839</r>
    <x>0.51833</x>
</link>

<flags>powerflow_only</flags>
</GridDyn>
```

This small XML files defines a two bus system shown in figure XXX. There are 5 sections to this model description. The first line describes the standard XML header information and is not used by GridDyn. The second line defines the simulation element and the name of the simulation. in general properties can be described in an either an element or as a property. There are certain aspects of parameters which can only be controlled in the element form but simple parameters either works fine. Capitalization of properties also does not matter. All object properties in GridDyn are represented by lower case strings, the xml reader converts all property names to lower case strings before input to GridDyn so capitalization doesn't

matter in the XML input. The property values themselves preserve capitalization and it is on a per property basis whether capitalization matters. For naming capitalization is preserved such that "object1" is distinct from "Object1". For this XML file the simulation is given the name 2bus_test. The version is for record keeping only and has not relevance to the simulation.

The second block defines a bus object with a name of "bus1" The bus is a slack bus indicated by `<bustype>SLK</bustype>` . Other options for this parameter include PQ, PV, SLK, afix. The angle and voltage are specified. A generator object is included. The element "generator" is recognized as a component and a new generator object is created with a name of "gen1". Finally a load is created with a name of "load1" and a fixed real power of 1.05 and a reactive power of 0.31.

```
<bus name="bus1">
   <bustype>SLK</bustype>
   <angle>0</angle>
   <voltage>1.05</voltage>
   <generator name="gen1">
   </generator>
   <load name="load1">
      <P>1.05</P>
      <Q>0.31</Q>
   </load>
</bus>
```

The second bus is defined in a similar way, except it does not define a bustype which means it defaults to a PQ bus. The Link is defined by

```
<link from="bus1" name="bus1_to_bus2" to="bus2">
   <b>0.127</b>
   <r>0.0839</r>
   <x>0.51833</x>
</link>
```

the properties b, r, and x are defined in the XML as elements. The "to" and "from" fields are specified using the names of the buses. These properties must be specified for the lines or the system will spit out a warning.

Finally the last two lines specify that the simulation should stop after a power flow. to add in dynamic modeling a few additional pieces of the XML can be added The Full dynamic simulation input file is shown in Listing7.1

```
<?xml version="1.0" encoding="utf-8"?>
<GridDyn name="2bus_test" version="1">

<bus name="bus1">
   <type>SLK</type>
   <angle>0</angle>
```

```
   <voltage>1.05</voltage>
   <generator name="gen1">
      <dynmodel>typical</dynmodel>
      <pmax>4</pmax>
   </generator>
   <load name="load1">
       <P>1.05</P>
       <Q>0.31</Q>
   </load>
</bus>

<bus name="bus2">
   <load name="load2">
      <P>0.45</P>
      <Q>0.2</Q>
      <event>@1|p=1.1</event>
   </load>
</bus>
<link from="bus1" name="bus1_to_bus2" to="bus2">
   <b>0.127</b>
   <r>0.0839</r>
   <x>0.51833</x>
</link>

<stoptime>10</stoptime>
<recorder period=0.05 field="auto">
   <file>twobusdynout.csv</file>
</recorder>

</GridDyn>
```

There are a few new sections in this file compared with the previous example. the code

```
<generator name="gen1">
   <dynmodel>typical</dynmodel>
   <pmax>4</pmax>
</generator>
```

now defines the generator to have a typical dynamic model, what this means will be detailed in the section on model parameters for specific models. It also specifies a Pmax value of 4 per unit. The line `<event>@1|p=1.1</event>` defines an event such at that at time 1.0 the p field of the load is set to 1.1 from the given value of 0.45. More details will be explained in the section on event specification. The final section

```
<stoptime>10</stoptime>
<recorder period=0.05 field="auto">
   <file>twobusdynout.csv</file>
</recorder>
```

set the simulation to run until a stoptime of 10 seconds. The recorder xml element defines a recorder to capture a set of automatic fields at a period of 0.05 seconds, and capture it to the file twobusdynout.csv upon completion of the scenario. More details on recorder specification are available later in this document.

7.2 Parameter Specification

simple parameters can be specified via elements or as attributes. Default units are in seconds for all times and time constants unless individual models assume differently. Power and impedance specifications are typically in PU values. Exceptions include "basepower" and "basevoltage" specifications which are in MW and KV respectively. The default units on any rates are in units per second. However, individual models are free to deviate from this standard as makes sense for them so check with the individual model type specification for details. Parameters in the XML can be specified in a number of different forms that are useful in different contexts. An example showing the various methods is in Listing 1

Listing 1: XML file for testing parameter input methods

```
<?xml version="1.0" encoding="utf-8"?>
<!--xml file to test parameter setting methods-->
<GridDyn name="input_tests" version="0.0.1">
<bus name="bus1">
   <load>
      <param name="P" value=0.4></param>
      <param field="q">0.3</param>
      <param field="ip" units="MW">55</param>
      <param>yq=0.11</param>
      <param name="iq(MW)" value=32/>
      <yp>0.5</yp>
   </load>

   <load yq=0.74 >
      <p units="puMW"> 0.31</p>
      <param>q(MW)=14.8</param>
      <param name="yp" unit="MW" value=127/>
   </load>
</bus>
</GridDyn>
```

The main variants involve varying how the units are placed. Units can be placed as an attribute named "unit" or "units" on the parameters either in a param element or and element named after the model parameter. They can also be placed in parenthesis at the end of the parameter name when the parameter name is a string contained in the elemental form. values can be places in a value element, as the content of an element, or following an equal sign when defined as a string like `<param>yq=0.11</param>`. Parameter assuming the default units are allowed to be placed as attributes of the object.

Listing 2: Example of a component XML element

```
<exciter name="ext1">
   <type>type1</type>
   <Aex>0</Aex>
   <Bex>0</Bex>
   <Ka>20</Ka>
   <Ke>1</Ke>
   <Kf>0.040</Kf>
   <Ta>0.200</Ta>
   <Te>0.700</Te>
   <Tf>1</Tf>
   <Urmax>50</Urmax>
   <Urmin>-50</Urmin>
</exciter>
```

7.3 Functions and Mathematical Operations

GridDyn XML input allows mathematical operators and expressions in any parameter specification, including complex expressions. Supported functions are shown in Tables 1 through 3. And supported operators are described in Table **??** Operators precedence is respected as are parenthesis. String operations are not supported but the definition system has features that support some use cases for string operations.

Table 1: Zero argument mathematical expressions

function	details
inf()	results in a large number (1e48) currently
nan()	uses nan("0")
pi()	pi
rand()	produces a uniform random number between 0 and 1
randn()	produces a normal random number with mean 0 and standard deviation of 1.0
randexp()	produces a random number from an exponential distribution with a mean of 1.0
randlogn()	produces a random number from a log normal distribution

Additionally most operators are supported including '+','-','*','/','^' and,'%' that do the appropriate operation. Operator precedence follows normal rules.

7.4 Component Description

components are defined in in elements matching the component name. For example describes an exciter component as part of a generator. The name attribute or element is common for all objects. A "description" can also be defined for all objects which is basically a string

Table 2: One argument mathematical expressions

function	details
sin(x)	sine of x
cos(x)	cosine of x
tan(x)	tangent of x
sinh(x)	hyperbolic sine of x
cosh(x)	hyperbolic cosine of x
tanh(x)	hyperbolic tangent of x
abs(x)	absolute value of x
sign(x)	return 1.0 if x>0 and -1.0 if x< 0 and 0 if x==0
asin(x)	arcsin of x
acos(x)	arccosine of x
atan(x)	arctangent of x
sqrt(x)	the square root of x
cbrt(x)	the cube root of x
log (x)	the natural logarithm of x $log(exp(x)) = x$
exp(x)	the exponential function e^x
log10(x)	the base 10 logarithm of x
log2(x)	the base 2 logarithm of x
exp2(x)	evaluates 2^x
ceil(x)	the smallest integer value such that ceil(x)>=x
floor(x)	the largest integer value such that floor(x)<=x
round(x)	the nearest integer value to x
trunc(x)	the integer portion of x
none(x)	return x
dec(x)	the decimal portion of x $trunc(x) + dec(x) = x$
randexp(x)	an exponential random variable with a mean of x

that can be added to any object. The "type" property is a keyword used to describe the detailed type of the component. In the above example the specific type of the component is "type1". GridDyn uses polymorphic objects for each of the components. The type defined in the XML file for each component defines the specific object to instantiate. If type is not specified the default type of the component is used.

Predefined components include

area , defines a region of the grid

bus the basic node of the system

link the basic object connecting buses together

relay primary object allowing control and triggers for other objects

sensor a form of a relay specifically targeted at sensing different parameters and allowing some direct signal processing on the measurements before output.

Table 3: Two argument mathematical expressions

function	details
atan2(x,y)	the 4 quadrant arctangent function
pow(x,y)	evaluates x^y
plus(x,y)	evaluates $x + y$
add(x,y)	evaluates $x + y$
minus(x,y)	evaluates $x - y$
subtract(x,y)	evaluates $x - y$
mult(x,y)	evaluates $x * y$
product(x,y)	evaluates $x * y$
div(x,y)	evaluate x/y
max(x,y)	return the greater of x or y
min(x,y)	returns the lesser of x or y
mod(x,y)	return the modulus of x and y e.g mod(5,3)=2
hypot(x,y)	evaluates $\sqrt{(x^2 + y^2)}$
rand(x,y)	return a random number between x and y
randn(x,y)	returns a random number from a normal distribution with mean x and variance y
randexp(x,y)	return a random number from an exponential distribution with mean x and variance y
randlogn(x,y)	returns a random number from a log normal distribution with mean x and variance y
randint(x,y)	returns a uniformly distributed random integer between x and y inclusive of x and y

load The basic consumer of energy

generator the basic producer of electricity

genmodel the dynamic model of a electrical generator

governor a generator governor

exciter an exciter for a generator

pss a power system stabilizer*

controlblock a basic control block

source A signal generator in GridDyn

simulation a simulation object

agc describe an automatic generation control*

reservedispatcher describe a reserve dispatcher*

scheduler a scheduling controller*

* these objects are in development and do not work consistently

Several component types of components are also defined that map onto the more general components, in some cases these define specific types in others they are simply maps. Examples include "fuse=>relay", "breaker=>relay", "transformer=>line", "block=>controlblock", "control=>relay", "tie=>line" custom definitions can also be defined if desired through a translate element.

7.5 Object Identification

there are many instances where it is necessary to identify an object for purposes of creating links or to extract a property or other. Internally there is a hierarchy of objects starting with the root simulation object. This allows a path like specification of the objects. There are 2 different notations for describing an object path, on based on colons, the other more similar to the URI specification for WEB like services,and both allow properties to be specified in a similar fashion. Objects can be referred to by 4 distinct patterns. The first is by the name of the object. The name should be unique within any given parent object. The second uses an object component name and index number for example "bus#0" would refer to the bus in index location 0, Using "bus!0" will also work the same as "bus#0" but can be used in settings where the '#' is not allowed. All indices are 0 based. The fourth makes use of the user id of an object, to use this objects would be identified by load$2 to locate the load with userID of 2. When searching for an object the system starts in the current directory for the search if it is not found it traverses to the root object and starts the search from there. An example of a specified path using the ':' notation is

```
area45::bus#6::load#0:p
```

the single colon marks that the final string represents a property. the same object in the URI notation would be

```
area45/bus#6/load#0?p
```

In some cases mixed notation might work but it is not recommended. The property indication can be left off when referencing an entire object. Starting the object identification string with and '@' or a beginning '/' implies searching starting in the root object, otherwise the search starts at whatever the current object of interest is.

7.6 special elements

In addition to the component elements several element names have special purposes.

7.6.1 translate

The translate element element is used to create a custom object definition

```
<translate name="special" component="exciter" type="type1"/>
```

putting this command in the xml file will allow objects using "special" as the element name instead of specifying "exciter" in the element name and a specific type. Translations unlike definitions are global and are only allowed in the root element of an XML file. If you wish to specify a default type for a component or other defined component translation the name or component can be left out.

7.6.2 definitions

The GridDyn XML file allows specification of definition strings that can be used as parameter values or in other definitions.

```
<define name="constant1" value=5/>
<define name="constant2" value=46 locked=1/>
<define name="constant3 value="constant1*constant2" eval=1/>
```

The above snippet of code defines 3 constants. Internally constants are stored as strings. If the eval attribute is specified the value string is evaluated before storing as a string, otherwise it will be stored as a string and evaluated on use. The locked attribute defines a global parameter, that cannot be overridden by another define command. The mechanisms allow programmatic or command line overrides of any internal definitions in the XML file. Inside the XML file they cross scope boundaries like a global variable. Regular definitions are only valid in elements they were defined in and subelements. So if a definition is used, define it in the root scope or it will only be applicable in subsection of the XML.

When using definitions they can be used as a variable in other languages wherever a string or numerical value would be used. They can also be used in string replacements like the following code

```
<bus name="bus_$#rowindex$_$#colindex$">
```

When evaluating the expression The parts of the string between the $ signs gets evaluated first in this case "#rowindex" and "#colindex" are part of an array structure which is defined in subsection **??**.

After the substring replacement, the entire expression is evaluated again for other definitions. There is a small set of predefined definitions including %date, %datetime, %time, which contain strings of the expected values at the time of file input.

7.6.3 custom

Translations are useful for readability, library elements allow duplication of objects with only minor modifications. For library elements the object is constructed once from XML and duplicated. Custom objects allow duplication of objects or sets of objects from the xml. A reference to the actual element source is stored and reprocessed at the time the custom object is encountered. This also allows a set of object to be defined in one input form to be imported by a different form and used to create objects described in the first. For instance you could create a library of different object sections and import that into another xml file and only use a few of the custom definitions you are interested in. Custom objects can use other custom objects but cannot define new custom sections. Custom objects can define a

set of required arguments and default values. When calling the custom element arguments can be defined. A brief example using custom elements is shown in Listing 3 and another using arguments is shown in Listing 4.

Listing 3: xml file using a custom defined object

```
<?xml version="1.0" encoding="utf-8"?>
<!--xml file to test custom elements-->
<griddyn name="customtest">
<custom name="cbus">
  <bus>
    <voltage>rand(0.95,1.05)</voltage>
  </bus>
</custom>
<array count=10>
  <cbus/>
</array>
</griddyn>
```

Listing 4: xml file using a custom defined object with arguments

```
<?xml version="1.0" encoding="utf-8"?>
<!--xml file to test custom elements-->
<griddyn name="customtest">
<custom name="busarray" args=1>
<array count="arg1">
  <bus>
    <voltage>rand(0.95,1.05)</voltage>
  </bus>
 </array>
</custom>
<busarray arg1=7/>
</griddyn>
```

7.6.4 configuration

A configuration element can define some parameters and operations for the XML reader itself. There are currently 2 parameters that can be specified "printlevel" and "match_type". The "printlevel" controls the verbosity of the output. The match type controls the default match capitalization for searching for specified fields. The field and valid values are in Table 4

7.6.5 event

Events are changes that take place during the course of the GridDyn simulation execution. They can be as simple as the example used previously or contain more complex specifications and multiple times and values. The events are likely to be updated significantly in the near

Table 4: reader configuration options

parameter	value	details
printlevel	none(0)	print only warnings and errors
	summary(1)	print only summary info
	detailed(2)	print detailed information
match_type	capital_case_match	match on specified, lower case, and upper case values (this is the default value)
	all	match to all cases [1ex]
	exact	only match to the given string. This option can be used to process files where every component and xml description field is in lower case in the xml to speed up processing slightly

future, and while much of the specification will remain the same some new capabilities will be in place. parameters for events are specified like those in the components. The available element or attribute names are shown in table 5

Table 5: Event Specification Options

parameter	description
target	the object to extract data from
field	the field of the target object to capture
time, t	The time the event is scheduled to occur
units	the desired units of the output
value	the value associated with the event
period	for periodic events set the period
file	the input file for a player type event
column	the column in a data file to use for the event

The field option of event specification is by far the most flexible. Any text directly in the event element is captured as the field.

7.6.6 Recorder

recorders are the primary data output system for GridDyn dynamic simulations. Like events recorders have a set of parameters associated with them. The details are in Table 6. Multiple recorder elements can be specified and the recorder for a single file can have multiple elements that get merged even if they are in different objects. They are keyed by recorder name and/or filename. Certain properties like the sampling period are specified on a recorder basis. Others are for the properties and data to record.

Recorder fields define which property of an an object to capture. This includes all properties and calculations involving properties. All functions and expressions defined in 7.3 are valid in recorder expressions.

Table 6: Recorder Specification Options

parameter	default	description
file	outputfile.csv	the file to save the data to
name	"recorder_#"	the name of the recorder for easy reference later
description		description that gets put in the header of the output file
column	-1	the column of the recorder in which to place the requested data
target		the object to extract data from
field		the field of the target object to capture
units	defUnit	the desired units of the output
offset	0	an offset index for a particular state
gain	1.0	a multiplier on the measurement
bias	0.0	a measurement bias
precision	7	the number of digits of precision to print for string formats
frequency	1.0	set the frequency of recording
period	1.0	set the measurement period
starttime	-inf	set a start time for the recorder
stoptime	inf	set a stop time for the recorder
autosave	0	set the recorder to save every N samples 0 for off
reserve	0	reserve space for N samples
period_resolution	0	set the minimum resolution for any time period (not usually user specified)

7.6.7 Solver

Solvers can be defined through the XML file. There are some default solvers defined but the solver element allows the definition of custom solvers applied to specific problem types. This allows specification of specific approximations or other configuration options for the solvers to use for solving various specific problems. Solver properties are shown in Table 7

Solvers have a set of options used to define what types of problems they are intended to solve. And another set of intended approximations. This information gets passed to the models whenever a solve is attempted. A listing of the possible modes is shown in Table 8. In some cases multiple modes can be combined in other cases they are mutually exclusive and the second will override the earlier specification. A number of approximation are also specified mainly targeting approximations to transmission lines. These approximations are suggestions rather than directives and models are free to ignore them. There are 3 independent approximations that can be used in various combinations and several descriptions which turn on the simplifications in a convenient form. Most approximations target the acline models, but future approximations can be added specifically looking at other models.

Table 7: Solver Control Options

parameter	default	description
printlevel	error(1)	may be specified with a string or number "debug"(2), "error"(1), "none"(0), "erro" only prints out error messages
approx	"none"	see Table 8 for details on possible options
flags		see Table 9 for details on available flags
tolerance	1e-8	the residual tolerance to use
name	solver_#	the name of the solver
index	automatic	the specified index of the solver
file		log file for the solver

Solvers can also include a number of options that are common across all solvers(though specific solvers may not implement them). Often specific solvers also include other options specific to that numerical solver.

7.6.8 import

Import statements are used to add an external file into the simulation. The file can be of type capable of being read by GridDyn. Import statements are typically single element statements thought they can have subelements if desired. A couple examples are shown in Listing 5.

Listing 5: Examples of import statements

```
<import>sep_lib.xml</import>
<import prefix="A1">subnetwork.csv</import>
<import final=true ext="xml">last_elements.odx</import>
```

The optional attributes/elements are described in table 10.

7.6.9 directory

The directory element allows the user to specify additional search paths for GridDyn to locate any files without an absolute path.

```
<directory>/home/usr/user1/GridDyn</directory>
<directory>
```

Table 8: Solver Modes

mode/approximation	description
local	used for local solutions
dae	solver is intended for solving a set of coupled differential-algebraic equations
differential	solver is intended to solve a set of coupled differential equations
algebraic	solver is intended to solve algebraic equations only
dynamic	solver is intended for dynamic simulations
powerflow	solver is intended for powerflow problems (implies !dynamic and algebraic)
extended	instructs the model to use an extended state formulation mainly targetted at state estimation problems
primary	opposite of extended
ac	solve for both voltage and angle on the buses
dc	solve only for the angle on AC buses, assume the voltage is fixed.
r, small_r	assume the resistance of transmission lines is small
small_angle	assume $sin(\theta) = \theta$
coupling	assume there is no coupling between the V and θ states
normal	use full detailed calculations
simple, simplified	use the small_r approximation
small_angle	use the small angle approximation
decoupled	use the coupling approximation
small_angle_decoupled	use the small angle and decoupled approximations
small_angle_simplified	use the samll angle and small r approximations
simplified_decoupled	use the small r and decoupling approximations
fast_decoupled	use all 3 approximations
linear	assume the problem is linear

Table 9: Solver Flags

mode	description
dense,sparse	set the solver to use a dense matrix solve or a sparse(default)
parallel,serial	set the solver to use parallel or serial(default) arrays, this is in the form of openMP array
constant_Jacobian	tell the solver to assume the Jacobian is constant
mask	tell the solver to use a masking element to shield specific variables from the solution. Thus functionality is used in some cases of initial condition generation and probably shouldn't be used externally

Table 10: import attributes

parameter	valid values	description
prefix	a string	a string to prefix all object names from the imported file
final	"true(1)"	if set to true the import is delayed until after all other non-final imports and the local file have been loaded
	"false(0)"	if set to false or not included the import is processed before any locally defined objects and in the order imports are specified
file	string	the file name, can also be interpreted from the element text
filetype	string	the extension to use for interpreting the import file if not specified the extension is determined from the file name
flags	ignore_step_up_transformers	the flags option is to add in additional options, it will likely be expanded as needed, currently the only options available is to ignore step up transformers in some formats of model input. As the file readers improve and become more integrated and consistent more options will be available.

7.6.10 library

GridDyn file input can include a library of predefined objects. This section is defined through a library element. Any of the components described above can be included as a library element. These library objects get stored in a separate holding area and are copied when any object uses a "ref" fields with a value of the library element name. The "ref" field can be either an element or an attribute. If type and "ref" are specified the type definition takes priority and the library object is cloned to the newly created object, if only "ref" is specified a new object is cloned directly from the library object. There can be multiple library sections, they simply get merged. By using import statements libraries can be defined in a separate file. A simple example using libraries and references is in Listing 6. The code describes 4 objects and generator model, an exciter and a governor, and a generator that uses the 3 previously defined submodels to make up thy dynamic components of the generator.

Listing 6: XML code example of a library element

```
<library>
   <model name="mod1">
      <type>fourthOrder</type>
      <D>0.040</D>
      <H>5</H>
      <Tdop>8</Tdop>
      <Tqop>1</Tqop>
      <Xd>1.050</Xd>
      <Xdp>0.350</Xdp>
      <Xq>0.850</Xq>
      <Xqp>0.350</Xqp>
   </model>
   <exciter name="ext1">
      <type>type1</type>
      <Aex>0</Aex>
      <Bex>0</Bex>
      <Ka>20</Ka>
      <Ke>1</Ke>
      <Kf>0.040</Kf>
      <Ta>0.200</Ta>
      <Te>0.700</Te>
      <Tf>1</Tf>
      <Urmax>50</Urmax>
      <Urmin>-50</Urmin>
   </exciter>
   <governor name="gov1">
      <type>basic</type>
      <K>16.667</K>
      <T1>0.100</T1>
      <T2>0.150</T2>
      <T3>0.050</T3>
   </governor>
```

```
    <generator name="gen1">
       <model ref="mod1"/>
       <exciter ref="ext1"/>
       <governor ref="gov1"/>
    </generator>
 </library>
```

Libraries are only allowed to be defined at the root object level, they are not allowed in any element that is a part of the root element so they are directly processed by the interpreter.

7.7 array

Arrays and if statement make up the control structures in the XM: file. Arrays allow objects and sets of objects to be generated in a loop,they can even contain other loops. An example file used for building some scalability tests is shown in Listing **??**. This file uses many of the concepts discussed previously.

Listing 7: xml file that shows scalability using arrays

```
<?xml version="1.0" encoding="utf-8"?>
<!--xml file to scalability using arrays-->
<griddyn name="test1" version="0.0.1">
<define name="garraySize" value=20/>
<define name="gcount" value="ceil(garraySize/3)" eval=1 />
<configuration>
<match_type>exact</match_type>
</configuration>
<library>
<generator name="default">
<P>3.8*(((garraySize^2)/(gcount^2))/9)</P>
<mbase>400</mbase>
 <exciter>
 <type>type1</type>
 <vrmin>-50</vrmin>
 <vrmax>50</vrmax>
 </exciter>
 <model/>
<governor/>
</generator>
<load name="addLoad">
     <Yp>0.5</Yp>
     <Yq>0.2</Yq>
     </load>
     <load name="constLd">
     <P>0.1</P>
     <Q>0.02</Q>
     <Ip>0.1</Ip>
```

```
     <Iq>0.02</Iq>
     <Yp>0.1</Yp>
     <Yq>0.02</Yq>
     </load>
</library>
   <array count=garraySize loopvariable="#rowindex">
   <array count=garraySize loopvariable="#colindex">
     <bus name="bus_$#rowindex$_$#colindex$">
    <load ref="constLd"/>
    </bus>
    </array>
    </array>
    <!--add in the additional loads -->

     <array start=1 stop=garraySize loopvariable="#rowindex" interval=2>
    <array start=1 stop=garraySize loopvariable="#colindex" interval=2>
     <bus name="bus_$#rowindex$_$#colindex$">
    <load ref="addLoad"/>
    </bus>
    </array>
    </array>
     <!--add in the generators -->
     <array start=1 stop=garraySize loopvariable="#rowindex" interval=3>
    <array start=1 stop=garraySize loopvariable="#colindex" interval=3>
     <bus name="bus_$#rowindex$_$#colindex$">
    <gen ref="default"/>
    <bustype>PV</bustype>
    <voltage>1.01</voltage>
    </bus>
    </array>
    </array>
    <!--add in the vertical links-->
    <array stop=garraySize loopvariable="#rowindex" start=2>
   <array count=garraySize loopvariable="#colindex">
     <link name="link_$#rowindex-1$_$#colindex$_to_$#rowindex$_$#colindex$">
    <r>0.001</r>
    <x>0.07</x>
    <from>bus_$#rowindex-1$_$#colindex$</from>
    <to>bus_$#rowindex$_$#colindex$</to>
    </link>
    </array>
    </array>

    <!--add in the horizonal links-->
    <array count=garraySize loopvariable="#rowindex">
   <array stop=garraySize loopvariable="#colindex" start=2>
     <link name="link_$#rowindex$_$#colindex-1$_to_$#rowindex$_$#colindex$">
```

```
    <r>0.001</r>
    <x>0.07</x>
    <from>bus_$#rowindex$_$#colindex-1$</from>
    <to>bus_$#rowindex$_$#colindex$</to>
    </link>
    </array>
    </array>
     <!--label the swing bus-->
     <busmodify name="bus_$1+3*floor(garraySize/6)$_$1+3*floor(garraySize/6)$">
     <bustype>SLK</bustype>
     <id>10000000</id>
     <voltage>1.03</voltage>
     </busmodify>

   <basepower>30</basepower>
   <timestart>0</timestart>
   <timestop>60</timestop>
   <timestep>0.02</timestep>
   <solver name="ida">
   <printlevel>1</printlevel>
   </solver>
</griddyn>
```

Arrays can have several attributes which define how the array is handled.

Table 11: array attributes

attribute	default	description
start	1	the index to start the array counter
stop	X	the last index to use, either stop or count must be specified
count	X	the number of loops, either stop or count must be specified
loopvariable	"#index"	the name of the definition to store the loop variable
interval	1.0	the interval between each iteration of the loop counter

7.7.1 if

If elements create a conditional inclusion. Most often used for conditional inclusion based on fixed parameters to allow a single file to do a few different scenarios. However, they can be tied in with random function generators and arrays to generate random distributions of elements. Any element component along with import and define statements are allowed in an if element. The if element must have an element or attribute named "condition". The condition is a string specifying a value or two values and a comparison operator. If

a single expression is given the elements in the if statement are evaluated as long as the expression does not result in a 0. Otherwise both sides of the expression are evaluated and the comparison is checked. If both sides evaluate to strings a string comparison is done, otherwise a numerical comparison if both sides result in numerical values. depending on the file type and reader '>' and '<' may need to be replaced with the XML character codes of > and < These codes are interpreted properly. Compound expressions are not yet supported. Eventually the goal will be to support conditions based on object values instead of values that can be evaluated in the element reader itself, but this capability is not yet allowed.

7.7.2 econ

The econ element describes data related to the costs and values of an object. It will be used for interaction with optimization solvers and the root object must be an optimization type simulation. While the element works fine, it doesn't do anything with the data.

7.7.3 position

A position element describes data related to the geophysical(or relative) position of an object. The element is ignored but will be further developed at a later time.

7.8 actions

The gridDynSimulation object can execute a number of types of actions. These can be controlled through the API but also through an action queue. The actions are defined and stored in a queue and executed when the run function is called. If no actions are defined some logic is in place to do something sensible, typically run a power flow then a dynamic simulation if dynamic components were instantiated. Actions allow a much finer grained control over this process. These actions can be loaded through the XML file eventually in a type of script(not enabled yet). actions are specified through an action element containing the action string. The string is translated into an action and stored in a queue.

```
<action>run 23.7</action>
```

The list of available command is shown in table 12, for all lines int the table (s) implies string parameter, (d) implies double parameter, (i) integer parameter, (X)* optional, (s|d|i), string or double or int*/ and for a given line everything following a # at the beginning of a word is considered a comment and ignored

8 Acknowledgments

Initial development of GridDyn was funded through lab directed research and development at Lawrence Livermore National Lab project 013-ERD-043). Additional support was provided through the DOE Advanced Grid Modeling program and through a joint project with the California Utilities of Pacific Gas and Electric, Southern California Edison, and San Diego Gas and Electric.

Table 12: GridDyn actions

action string	description
ignore XXXXXX	do nothing
set parameter(s) value(d)	set a particular parameter; the parameter can include an object path
setall objecttype(s) parameter(s) value(d)	set a parameter on all objects of a particular type
setsolver mode(s) solver(s|i)	set the solver to use for a particular mode of operation
settime newtime(d)	set the simulation time
print parameter(s) setstring(s)	print a parameter (can include path) using the string definition in setstring
powerflow	run a powerflow
step solutionType(s)*	take a single step of the specified solution type
eventmode stop(d)* step(d)*	run in event driven power flow mode until stop with step
initialize	run the initialization routine
dynamic solutionType(s)* stop(d)* step(d)*	run a dynamic simulation
dynamicdae stop(d)*	run a dynamic simulation using DAE solver
dynamicpart stop(d)* step(d)*	run a dynamic simulation using the partitioned solver
dynamicdecoupled stop(d)* step(d)*	run a dynamic simulation using the decoupled solver
reset level(i) reset the simulation to the specified level	
iterate interval(d)* stop(d)*	run an iterative power flow with the given interval
run time(d)*	run the simulation using the default mode to the given time
save subject(s) file(s)	save a particular type of file
load subject(s) file(s)	load a particular type of file
add addstring(s)	add something to the simulation
rollback point(s|d)	rollback to a saved checkpoint (not implemented yet)
checkpoint name(s)	save a named checkpoint (not implemented yet)
contingency ????	run a contingency analysis (not implemented yet)
continuation ????	run a continuation analysis (not implemented yet)

9 Contributers

Lead Author: Philip Top
contributers

- Steve Smith
- Carol Woodward
- Eddy Banks
- Liang Min
- Brian Kelley
- Rafael Rivera-Soto
- Yining Qin

past contributors

- Brett Robbins
- Malquan Gaillard
- Jaleel King

References

[1] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward, "SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers," *ACM Trans. Math. Softw.*, vol. 31, no. 3, pp. 363–396, 2005.

A Settable Object Properties

Appendix A describes the parameters for each of the models present in GridDyn as of Version 0.5. The tables are automatically generated via Matlab scripts so there are a few bugs and some missing information as of yet. Each table has 4 columns. The first column specifies the string or strings that can be used to set this property, multiple strings that do the same thing are separated by a comma. The second columns defines the type of parameter, number implies a numeric value, string implies a string field, and flag is a flag or boolean variable which can be set to true with "true", or any number greater than 0.1 (typically 1), and set to false for any number less than 0.1 or "false". The third column lists the default value if applicable and the fourth column is a description. In many cases the default units will be described in [] at the beginning of the description, the default units are the units of the default and the unit that is assumed if no units are given to the set command. All the set functions cascade to parent classes which are identified in the table captions.

string(s)	type	default	description
updateperiod, period	number	1e+48	the update period
updaterate, rate	number	0	
nextupdatetime	number	1e+48	the next scheduled update
basepower	number	100	the base power of the object
enabled	number	0	
number	number	0	
renumber	number	0	
id, id	number		a user defined id for the object
name	string		the text name of the object
rename	string	0	
description	string		storage for a description of the object meant for user, no operational impact
enable, status	flag	0	
disabled	flag	0	
searchable	flag	0	

Table A1: Set options for gridCoreObject objects.

string(s)	type	default	description
enabled	number	0	
status	number	0	
connected	number	0	
basepower	number	100	the base power of the object
basefreq, basefrequency	number	376.9911	the base frequency of the system default to 60Hz
status	string	0	
flags	string	0	
error	flag	0	
late_b_initialize	flag	0	
no_gridobject_set	flag	0	

Table A2: Set options for gridObject objects. See Table A1 for additional settable options through gridCoreObject

string(s)	type	default	description

Table A3: Set options for gridPrimary objects. See Table A2 for additional settable options through gridObject

string(s)	type	default	description

Table A4: Set options for gridSecondary objects. See Table A2 for additional settable options through gridObject

string(s)	type	default	description

Table A5: Set options for gridSubModel objects. See Table A2 for additional settable options through gridObject

string(s)	type	default	description
basepower	number	100	the base power of the object
basefrequency, basefreq	number	376.9911	the base frequency of the system default to 60Hz
reverse_converge	flag	0	
direction_oscillate	flag	0	

Table A6: Set options for gridArea objects. See Table A3 for additional settable options through gridPrimary

string(s)	type	default	description
voltage, vol	number	1	[p.u.] per unit voltage magnitude
angle, ang	number	0	[rad] voltage angle
basevoltage, base vol	number	120	[kV] base voltage level
basepower	number	100	the base power of the object
basefrequency, basefreq	number	376.9911	the base frequency of the system default to 60Hz
disconnect	number	0	
p, gen p	number	0	
q, gen q	number	0	
load p	number	0	
load q	number	0	
shunt g	number	0	
g	number	0	
shunt b	number	0	
b	number	0	
zone	number	1	zone control for some reporting purposes for labeling purposes
status	string	0	
connected	flag	0	

Table A7: Set options for gridBus objects. See Table A3 for additional settable options through gridPrimary

string(s)	type	default	description
tolerance, rtol	number	0	
voltagetolerance	number	0	
angletolerance	number	0	
defaulttolerance	number	0	
tolerancerelaxation	number	0	
powerflowstarttime	number	NaN	power flow start time if nullval then it computes based on start time;
timetolerance	number	0	
poweradjustthreshold	number	0.01	tolerance on the power adjust step

maxpoweradjustiterations	number	0
defpowerflow	number	0
defdae	number	0
defdynalg	number	0
defdyndiff	number	0
maxvoltageadjustiterations	number	0
powerflowfile	string	0
defpowerflow	string	0
defdae	string	0
defdynalg	string	0
defdyndiff	string	0
action	string	0
ordering	string	0
dynamicsolvermethod	string	0
threads	flag	0

Table A9: Set options for gridDynSimulation objects. See Table A8 for additional settable options through gridSimulation

string(s)	type	default	description
voltage, vol	number	1	[p.u.] per unit voltage magnitude
angle, ang	number	0	[rad] voltage angle
basefrequency, basefreq	number	376.9911	the base frequency of the system default to 60Hz
vtarget	number	1	a target voltage
atarget	number	0	an angle Target(for SLK and afix bus types)
qmax	number	0	
qmin	number	0	
pmax	number	0	
pmin	number	0	
vmax	number	1e+48	[p.u.] voltage maximum
vmin	number	0	[p.u.] voltage minimum
autogenp	number	0	
autogenq	number	0	
autogendelay	number	0	
voltagetolerance, vtol	number	-1	voltage tolerance
angletolerance, atol	number	-1	angle tolerance
tw	number	0.1	time constant for the frequency estimator
lowvdisconnect	number	0	
type	string	0	
bustype	string	0	
pflowtype	string	0	
dyntype	string	0	

status	string	0
compute_frequency	flag	0

Table A10: Set options for acBus objects. See Table A7 for additional settable options through gridBus

string(s)	type	default	description
pset	number	-1e+48	[p.u.] target power set point
qmax	number	1e+48	[pu mbase] max steady state reactive power values for Power flow analysis
qmin	number	-1e+48	[pu mbase] min steady state reactive power values for Power flow analysis
adjustment	number	0	
xs	number	1	generatore impedence defined on Mbase;
rs	number	0	the real part of the generator impedance
eft	number	0	place to store a constant the exciter field
vref, vtarget	number	-1	voltage target for the generator at the control bus
rating, base, mbase	number	100	MW the internal base power of the generator;
dpdt	number	0	define the power ramp
dqdt	number	0	define the reactive power ramp
basepower	number	100	the base power of the object
basevoltage	number	120	[V] base voltage
basefrequency, basefreq	number	376.9911	the base frequency of the system default to 60Hz
participation	number	1	[%]a participation factor used in auto allocating load.
vcontrolfrac, vregfraction, vcfrac	number	1	[%] fraction of output reactive power to maintain voltage regulation
pmax	number	1e+48	[pu mbase]max steady state real power values for the generator
pmin	number	-1e+48	[pu mbase] min steady state real power values for the generator
capabiltycurve	number	0	
remote	number	0	
dynmodel	string	0	
remote	string	0	
remote_power_control	string	0	
p	string	0	[p.u.] Electrical generation real power output

q	string	0	[p.u.] Electrical generation reactive power output
capabiltycurve	flag	0	
variable	flag	0	
variablegen	flag	0	
reserve	flag	0	
reservecapable	flag	0	
agc	flag	0	
agccapble	flag	0	
indirect_voltage_control	flag	0	

Table A13: Set options for gridDynGenerator objects. See Table A4 for additional settable options through gridSecondary

string(s)	type	default	description
timestart, start, starttime	number	0	[s] start time
abstime, walltime	number	0	[s] seconds in unix time of the system start time;
stoptime, stop, timestop	number	0	[s] end time
consoleprintlevel, consoleprintlevel	number	3	logging level for printing to the console
logprintlevel, logprintlevel	number	3	logging level for saving to a file (if a file was specified)
steptime, step, timestep	number	0.05	[s] time step
minupdatetime	number	0.0001	minimum time period to go between updates; for the hybrid simultaneous partitioned solution
maxupdatetime	number	1e+48	(s) max time period to go between updates
staterecordperiod	number	-1	how often to record the state
recordstop	number	1e+48	[s] recorder stop time
recordstart	number	-1e+48	[s] recorder start time
recorddirectory	string		folder location for storing recorded files
printlevel	string	0	
logfile	string		log file name
statefile	string		record file for the state
sourcefile	string		main source file name

Table A8: Set options for gridSimulation objects. See Table A6 for additional settable options through gridArea

string(s)	type	default	description

Table A11: Set options for dcBus objects. See Table A10 for additional settable options through acBus

string(s)	type	default	description
dvdt	number	0	ramp rate for voltage
dfdt	number	0	ramp rate for frequency

Table A12: Set options for infiniteBus objects. See Table A7 for additional settable options through gridBus

string(s)	type	default	description
vcutout	number	-1	
vmax	number	1e+48	

Table A14: Set options for variableGenerator objects. See Table A13 for additional settable options through gridDynGenerator

string(s)	type	default	description
load p	number	0	[p.u.] real component of the load (constant Power)
load q	number	0	[p.u.] imaginary component of the load (constant Power)
yp, shunt g, zr	number	0	[p.u.] the impedance load in MW
yq, shunt b, zq	number	0	[p.u.] the reactive impedance load in MVar
ir, ip	number	0	[p.u.] real current; (constant current)
iq	number	0	[p.u.] imaginary current (constant current)
pf	number	0	
powerfactor	number	0	
qratio	number	0	power factor multiply sqrt((1-pf*pf)/pf*pf)
usepowerfactor	number	0	
converttoimpedance	number	0	
vpqmin	number	0.7	low voltage at which the PQ powers convert to an impedance type load
vpqmax	number	1.3	upper voltage at which the PQ powers convert to an impedance type load
pqlowvlimit	number	0	
basevoltage, base vol	number		base voltage of the load
basepower, base pow	number	100	the base power of the object
usepowerfactor	flag	0	
converttoimpedance	flag	0	
no_pqvoltage_limit	flag	0	

Table A15: Set options for gridLoad objects. See Table A4 for additional settable options through gridSecondary

string(s)	type	default	description
consume	number	0	
fraction	number	0	
subload	string	0	
fraction	string	0	

Table A16: Set options for compositeLoad objects. See Table A15 for additional settable options through gridLoad

string(s)	type	default	description

Table A17: Set options for gridRampLoad objects. See Table A15 for additional settable options through gridLoad

string(s)	type	default	description
spread, band	number	0.01	the voltage spread to use when calculating the parameters
bounds	number	0	
usebounds	number	0	
mult, multiplier	number	1	a load multiplier
detail	number	0	
dual	number	0	
dualmode	number	0	
lineartriple	number	0	
detail	string	0	
mode	string	0	
coupling	string	0	
dyncoupling	string	0	
pflow	string	0	
pflowcoupling	string	0	

Table A18: Set options for gridLabDLoad objects. See Table A17 for additional settable options through gridRampLoad

string(s)	type	default	description
min_t	number	0	
max_t	number	100	
min_l	number	0	
max_l	number	0	
mean_t	number	0	
mean_l	number	0	
scale_t	number	0	
stdev_l	number	0	
interpolate	number	0	
repeated	number	0	
independent	number	0	
proportional	number	0	
seed	number	0	
trigger_dist	string	0	
time_dist	string	0	
size_dist	string	0	
change_dist	string	0	
independent	flag	0	

interpolate	flag	0
step	flag	0
repeated	flag	0
proportional	flag	0

Table A24: Set options for gridRandomLoad objects. See Table A17 for additional settable options through gridRampLoad

string(s)	type	default	description
r1	number	0.05	primary resistance on the motor
x1	number	0.15	primary inductance of the motor
xm	number	5	the inductive load of the motor
alpha	number	1	alpha parameter for torque conversion
beta	number	0	beta parameter for torque conversion
gamma	number	0	gamma parameter for torque conversion
base, mbase, rating	number	-100	system machine base
Vcontrol	number	1	whether the motor has some voltage controls for tweaking power (basically a transformer attached motor

Table A19: Set options for motorLoad objects. See Table A15 for additional settable options through gridLoad

string(s)	type	default	description
rs	number	1e+48	[p.u.] resistive load (constant impedance)

Table A20: Set options for motorLoad3 objects. See Table A19 for additional settable options through motorLoad

string(s)	type	default	description
r2	number	0.002	3rd level loop resistance
x2	number	0.04	3 impedence loop reactance

Table A21: Set options for motorLoad5 objects. See Table A20 for additional settable options through motorLoad3

string(s)	type	default	description
a, amplitude	number	0	
period, pulseperiod	number	1e+48	
frequency, freq, pulsefreq	number	0	
dutycycle	number	0.5	
shift	number	0	storage for phase shift fraction (should be between 0 and 1)
base, baseload, baseloadp, p	number	0	
baseloadq, q	number	0	
transition_time, transtime, transition	number	0.05	
invert	number	0	
type	string	0	
pulsetype	string	0	
cosine	string	0	

Table A22: Set options for gridPulseLoad objects. See Table A15 for additional settable options through gridLoad

string(s)	type	default	description
a, amplitude, amp	number	0	
frequency, freq	number	0	
period, sineperiod	number	1e+48	
phase	number	0	
dfdt	number	0	
dadt	number	0	
pulsed	number	0	

Table A23: Set options for gridSineLoad objects. See Table A22 for additional settable options through gridPulseLoad

string(s)	type	default	description
scalefactor, scaling	number	1	
qratio	number	NaN	
filename, file	string	0	
units	string	0	
mode	string	0	
timemode	string	0	
absolute	flag	0	
relative	flag	0	
step	flag	0	
interpolate	flag	0	

Table A25: Set options for gridFileLoad objects. See Table A17 for additional settable options through gridRampLoad

string(s)	type	default	description
alphap, ap	number	0	
alphaq, aq, alpha, a	number	0	

Table A26: Set options for exponentialLoad objects. See Table A15 for additional settable options through gridLoad

string(s)	type	default	description
betap	number	0	
betaq, beta	number	0	
loadtype	string	0	

Table A27: Set options for fDepLoad objects. See Table A26 for additional settable options through exponentialLoad

string(s)	type	default	description
qlow	number	0	the lowest available Q block level
qhigh	number	1e+48	the maximum reactive power block level
qmin	number	-1e+48	the minimum reactive power
qmax	number	1e+48	the maximum reactive power output
vmax	number	1.2	the high voltage threshold
vmin	number	0.8	the low voltage threshold
yq	number	0	
step	number	0	
participation	number	1	a participation factor
block	number	0	
count	number	0	
blocks	string	0	
block	string	0	
mode	string	0	
control	string	0	

Table A28: Set options for svd objects. See Table A17 for additional settable options through gridRampLoad

string(s)	type	default	description
state	number	0	
switch	number	0	
switch1	number	0	
breaker	number	0	
breaker_open	number	0	
breaker1	number	0	
breaker_open1	number	0	
switch2	number	0	
breaker2	number	0	
breaker_open2	number	0	
pset	number	0	the scheduled power of the link
loss, lossfraction	number	0	the fraction of power transferred that is lossed
ratinga, rating	number	-1e+48	the long term rating of the link
ratingb	number	-1e+48	the short term rating of the link
ratinge, emergency_rating, erating, ratingc	number	-1e+48	the emergency rating of the link
curcuit	number	0	
zone	number	1	publically accessible loss zone indicator not used internally
bus1	string	0	
from	string	0	
bus2	string	0	

to	string	0
status	string	0

Table A29: Set options for gridLink objects. See Table A3 for additional settable options through gridPrimary

string(s)	type	default	description
tap	number	1	tap position, neutral t = 1;
tapangle	number	0	[deg] phase angle for phase shifting transformer
no_pflow_control	number	0	
no_pflow_adjustments	number	0	
cbus	number	0	
controlbus	number	0	
vmin	number	0	minimum voltage before changing the tap
vmax	number	1e+48	maximum voltage before changing the tap
vtarget	number	-1e+48	target voltage
pmin	number	-1e+48	the minimum power level before changing the tap
pmax	number	1e+48	the maximum power before changing the tap
ptarget	number	-1e+48	the target power flow
qmin, min	number	-1e+48	the maximum power before changing the tap
qmax, max	number	1e+48	the minimum power level before changing the tap
qtarget, target	number	-1e+48	the target reactive power flow
direction	number	1	variable storing whether the directional derivate of the tap changes with respect to voltage or power is positive or negative
mintap	number	0.9	minimum tap setting
maxtap	number	1.1	maximum tap setting
mintapangle	number	-0.7854	minimum tap angle
fault	number	0	
maxtapangle	number	0.7854	maximum tap angle
stepsize, tapchange	number	0.01	step size of the adjustment for non-continuous adjustments
nsteps	number	0	
dtapdt	number	0	rate of change of the tap
dtapadt	number	0	rate of change of the tapAngle
controlmode	string	0	
mode	string	0	

control_mode	string	0	
change	string	0	
change_mode	string	0	
changemode	string	0	
stepmode	string	0	
center	string	0	
center_mode	string	0	
centermode	string	0	
bus, controlbus	string		the control bus and number setting are not fully determined until initialization so this stores information from the startup phase

Table A31: Set options for adjustableTransformer objects. See Table A30 for additional settable options through acLine

string(s)	type	default	description
length	number	0	[km] transmission line length
tap, ratio	number	1	tap position, neutral t = 1;
tapangle	number	0	[deg] phase angle for phase shifting transformer
fault	number	0	
minangle	number	-1.5708	the minimum angle of the link can handle
maxangle	number	1.5708	the maximum angle the link can handle–related to rating
approximation	string	0	

Table A30: Set options for acLine objects. See Table A29 for additional settable options through gridLink

string(s)	type	default	description

Table A32: Set options for acdcConverter objects. See Table A29 for additional settable options through gridLink

string(s)	type	default	description

Table A33: Set options for dcLink objects. See Table A29 for additional settable options through gridLink

string(s)	type	default	description
basepower	number	100	the base power of the object
basefrequency, basefreq	number	376.9911	the base frequency of the system default to 60Hz
terminals	number	0	
bus	string	0	
from	string	0	
to	string	0	
connection	string	0	

Table A34: Set options for subsystem objects. See Table A29 for additional settable options through gridLink

string(s)	type	default	description
r	number	0	
x	number	0	
pset	number	0	
pout	number	0	
from	string	0	
to	string	0	

Table A35: Set options for hvdc objects. See Table A34 for additional settable options through subsystem

string(s)	type	default	description
segmentationlength, segmentlength	number	50	the length of each segment
length	number	0	
fault	number	-1	

Table A36: Set options for longLine objects. See Table A34 for additional settable options through subsystem

string(s)	type	default	description

Table A37: Set options for rectifier objects. See Table A29 for additional settable options through gridLink

string(s)	type	default	description
status	number	0	
status	string	0	

Table A38: Set options for zBreaker objects. See Table A29 for additional settable options through gridLink

string(s)	type	default	description
val, setval, level, value	number	0	the output corresponding to the last setTime
type, sourcetype	string		string for use by applications to indicate usage

Table A39: Set options for gridSource objects. See Table A5 for additional settable options through gridSubModel

string(s)	type	default	description
dodt, ramp, rate	number	0	the ramp rate of the output

Table A40: Set options for rampSource objects. See Table A39 for additional settable options through gridSource

string(s)	type	default	description
a, amplitude	number	0	pulse amplitude
period	number	1e+48	pulse period
dutycycle	number	0.5	pulse duty cycle
shift	number	0	storage for phase shift fraction (should be between 0 and 1)
base, set, output	number		the base level of the output
invert	number	0	
type	string	0	
pulsetype	string	0	

Table A41: Set options for pulseSource objects. See Table A39 for additional settable options through gridSource

string(s)	type	default	description
a, amplitude, amp	number	0	
frequency	number	0	
phase	number	0	
dfdt	number	0	
dadt	number	0	
pulsed	number	0	

Table A42: Set options for sineSource objects. See Table A41 for additional settable options through pulseSource

string(s)	type	default	description
min_t	number	0	the minimum time between random updates
max_t	number	100	the maximum time between random updates
min_l	number	0	the minimum level of the update
max_l	number	0	the maximum change
mean_t	number	0	the mean time between changes
mean_l	number	0	the mean level change
scale_t	number	0	scaling factor for the times
stdev_l	number	0	the standard deviation of the level changes
interpolate	number	0	
repeated	number	0	
proportional	number	0	
seed	number	0	
trigger_dist	string	0	
time_dist	string	0	
size_dist	string	0	
change_dist	string	0	
interpolate	flag	0	
step	flag	0	
repeated	flag	0	
proportional	flag	0	

Table A43: Set options for randomSource objects. See Table A40 for additional settable options through rampSource

string(s)	type	default	description
filename	string	0	
file	string	0	
flags	string	0	
mode	string	0	

Table A44: Set options for fileSource objects. See Table A40 for additional settable options through rampSource

string(s)	type	default	description
k, gain	number	1	gain
bias, b	number	0	bias
omax, max, limit	number	1e+48	max output value
omin, min	number	-1e+48	min output value
rampmax, ramplimit	number	1e+48	rate of change max value
rampmin	number	-1e+48	rate of change min value
resetlevel	number	-0.001	the level below or above the max/min that the limiters should be removed
use_limits	flag	0	
simplified	flag	0	
differential_input	flag	0	
use_ramp_limits	flag	0	

Table A45: Set options for basicBlock objects. See Table A5 for additional settable options through gridSubModel

string(s)	type	default	description
iv, initial_value	number	0	the initial value(current value) of the integral
t	number	0	

Table A46: Set options for integralBlock objects. See Table A45 for additional settable options through basicBlock

string(s)	type	default	description
t1, t	number	0.1	the time constant

Table A47: Set options for delayBlock objects. See Table A45 for additional settable options through basicBlock

string(s)	type	default	description
t1, t	number	0.1	delay time constant for the derivative filtering operation

Table A48: Set options for derivativeBlock objects. See Table A45 for additional settable options through basicBlock

string(s)	type	default	description
t1, t	number	0.1	delay time constant
t2	number	0	upper time constant

Table A49: Set options for controlBlock objects. See Table A45 for additional settable options through basicBlock

string(s)	type	default	description
level, dblevel, deadbandlevel	number	0	the output level while the input is inside the deadband
deadband	number	0	
db	number	0	
deadbandhigh, dbhigh, high	number	-1e+48	upper limit on the deadband
deadbandlow, dblow, low	number	1e+48	lower deadband limit
rampband, ramp, rampupband, rampup	number	0	ramp band on the up side
rampdownband, rampdown	number	0	ramp band on the low side
resetlevel	number	0	
reset	number	0	
resethigh	number	-1e+48	the reset level to go off the deadband
resetlow	number	1e+48	the reset level to go back in the deadband on the low side
shifted	flag	0	
unshifted	flag	0	
no_down_deadband	flag	0	

Table A50: Set options for deadbandBlock objects. See Table A45 for additional settable options through basicBlock

string(s)	type	default	description
vref	number	1	[p.u.] reference voltage for voltage regulator
ka	number	10	[p.u.] amplifier gain
ta	number	0.004	[s] amplifier time constant
vrmax, urmax	number	6	[p.u.] upper voltage limit
vrmin, urmin	number	-5.1	[p.u.] lower voltage limit
vbias	number	0	bias field level for adjusting the field output so the ref can remain at some nominal level

Table A51: Set options for gridDynExciter objects. See Table A5 for additional settable options through gridSubModel

string(s)	type	default	description
ke	number	1	
te	number	1	
kf	number	0.03	
tf	number	1	
aex	number	0	
bex	number	0	

Table A52: Set options for gridDynExciterIEEEtype1 objects. See Table A51 for additional settable options through gridDynExciter

string(s)	type	default	description
tf1	number	1	
tf2	number	1	

Table A53: Set options for gridDynExciterIEEEtype2 objects. See Table A52 for additional settable options through gridDynExciterIEEEtype1

string(s)	type	default	description
tb	number	1	
tc	number	0	

Table A54: Set options for gridDynExciterDC1A objects. See Table A52 for additional settable options through gridDynExciterIEEEtype1

string(s)	type	default	description

Table A55: Set options for gridDynExciterDC2A objects. See Table A54 for additional settable options through gridDynExciterDC1A

string(s)	type	default	description
xd, xs	number	1.05	[p.u.] d-axis reactance
rs	number	0	[p.u.] generator resistance
base, mbase	number	100	[p.u.] the operating base of the generator

Table A56: Set options for gridDynGenModel objects. See Table A5 for additional settable options through gridSubModel

string(s)	type	default	description
xd	number	1.05	[p.u.] d-axis reactance
rs	number	0	[p.u.] generator resistance
base	number	100	[p.u.] the operating base of the generator
kw	number	13	speed gain for the damping system

Table A57: Set options for gridDynGenModelClassical objects. See Table A56 for additional settable options through gridDynGenModel

string(s)	type	default	description
k	number	16.667	[p.u.] droop gain (1/R)
r	number	0	
t1	number	0.1	[s] droop control time constant 1
t2	number	0	[s] droop control time constant 2
t3	number	0	[s] throttle response
omegaref, wref	number	-1e+48	[rad] reference frequency
pmax	number	1e+48	[p.u.] maximum turbine output
pmin	number	-1e+48	[p.u.] minimum turbine output
deadband, deadbandhigh	number	-1e+48	upper threshold on the deadband;
deadbandlow	number	1e+48	lower threshold on the deadband;

Table A58: Set options for gridDynGovernor objects. See Table A5 for additional settable options through gridSubModel

string(s)	type	default	description

Table A59: Set options for gridDynPSS objects. See Table A5 for additional settable options through gridSubModel

string(s)	type	default	description
p, proportional	number	1	proportional control constant
i, integral	number	0	integral control constant
d, derivative	number	0	differential control constant
t, t1	number	0.01	filtering delay on the input for the differential calculation
iv, initial_value	number	0	intermediate value for calculations

Table A60: Set options for pidBlock objects. See Table A45 for additional settable options through basicBlock

string(s)	type	default	description
t1	number	0.1	delay time constant for the derivative filtering operation
t2	number	0.1	filter on the derivative of block 1

Table A61: Set options for filteredDerivativeBlock objects. See Table A45 for additional settable options through basicBlock

string(s)	type	default	description
gain	number	1	extra gain factor
arg	number	0	second argument for 2 argument functions
function	string	0	
func	string	0	

Table A62: Set options for functionBlock objects. See Table A45 for additional settable options through basicBlock

string(s)	type	default	description
lut	string	0	
element	string	0	
file	string	0	

Table A63: Set options for lutBlock objects. See Table A45 for additional settable options through basicBlock

string(s)	type	default	description
a	string	0	
b	string	0	

Table A64: Set options for transferFunctionBlock objects. See Table A45 for additional settable options through basicBlock

string(s)	type	default	description
xd, xs	number	1.05	[p.u.] d-axis reactance
maxangle	number	1.5533	maximum firing angle
minangle	number	-1.5533	minimum firing angle
rs	number	0	[p.u.] generator resistance

Table A65: Set options for gridDynGenModelInverter objects. See Table A56 for additional settable options through gridDynGenModel

string(s)	type	default	description
x, xq	number	0.85	[p.u.] q-axis reactance
xl	number	0	[p.u.] leakage reactance
xp, xdp	number	0.35	[p.u.] d-axis transient reactance
tdop, td0p, top, t0p	number	8	[s] d-axis time constant

Table A66: Set options for gridDynGenModel3 objects. See Table A57 for additional settable options through gridDynGenModelClassical

string(s)	type	default	description
xd	number	1.05	[p.u.] d-axis reactance
xqp	number	0.35	[p.u.] q-axis transient reactance
tqop, tq0p, top, t0p	number	1	[s] q-axis time constant
s1, s10	number	1	the saturation S (1.0) const
s12	number	1	the saturation S(1.2)
saturation_type	string	0	

Table A67: Set options for gridDynGenModel4 objects. See Table A66 for additional settable options through gridDynGenModel3

string(s)	type	default	description
tqopp, tq0pp	number	0.1	
taa	number	0	
tdopp, td0pp	number	0.8	
xdpp, xpp	number	0.175	
xqpp	number	0.175	

Table A68: Set options for gridDynGenModel5 objects. See Table A67 for additional settable options through gridDynGenModel4

string(s)	type	default	description

Table A69: Set options for gridDynGenModel5type2 objects. See Table A68 for additional settable options through gridDynGenModel5

string(s)	type	default	description

Table A70: Set options for gridDynGenModel5type3 objects. See Table A66 for additional settable options through gridDynGenModel3

string(s)	type	default	description

Table A71: Set options for gridDynGenModel6 objects. See Table A68 for additional settable options through gridDynGenModel5

string(s)	type	default	description

Table A72: Set options for gridDynGenModel6type2 objects. See Table A69 for additional settable options through gridDynGenModel5type2

string(s)	type	default	description

Table A73: Set options for gridDynGenModelGENROU objects. See Table A68 for additional settable options through gridDynGenModel5

string(s)	type	default	description

Table A74: Set options for gridDynGenModel8 objects. See Table A71 for additional settable options through gridDynGenModel6

string(s)	type	default	description
t3	number		[s] servo motor time constant
pup, ramplimit	number		[p.u.] upper ramp limit
pdown	number		[p.u.] lower ramp limit

Table A75: Set options for gridDynGovernorIeeeSimple objects. See Table A58 for additional settable options through gridDynGovernor

string(s)	type	default	description
ts, t1	number	0.1	[s] droop control time constant 1
tc, t2	number	0	[s] droop control time constant 2
t3	number		[s] Transient gain time constant
t4	number		[s] Power fraction time constant
t5	number		[s] Reheat time constant

Table A76: Set options for gridDynGovernorReheat objects. See Table A58 for additional settable options through gridDynGovernor

string(s)	type	default	description
dt	number	0	speed damping constant

Table A77: Set options for gridDynGovernorTgov1 objects. See Table A75 for additional settable options through gridDynGovernorIeeeSimple

string(s)	type	default	description
k	number	16.667	[p.u.] droop gain (1/R)
t1	number	0.1	[s] droop control time constant 1
t2	number	0	[s] droop control time constant 2
t3	number		[s] servo motor time constant
pup	number		[p.u.] upper ramp limit
pdown	number		[p.u.] lower ramp limit
pmax	number	1e+48	[p.u.] maximum turbine output
pmin	number	-1e+48	[p.u.] minimum turbine output

Table A78: Set options for gridDynGovernorHydro objects. See Table A75 for additional settable options through gridDynGovernorIeeeSimple

string(s)	type	default	description
k	number	16.667	[p.u.] droop gain (1/R)
t1	number	0.1	[s] droop control time constant 1
t2	number	0	[s] droop control time constant 2
t3	number		[s] servo motor time constant
pup	number		[p.u.] upper ramp limit
pdown	number		[p.u.] lower ramp limit
pmax	number	1e+48	[p.u.] maximum turbine output
pmin	number	-1e+48	[p.u.] minimum turbine output

Table A79: Set options for gridDynGovernorSteamNR objects. See Table A75 for additional settable options through gridDynGovernorIeeeSimple

string(s)	type	default	description

Table A80: Set options for gridDynGovernorSteamTCSR objects. See Table A79 for additional settable options through gridDynGovernorSteamNR

string(s)	type	default	description
samplingrate	number	0	
ts	number	0	
commid	number	0	
commdestid	number	0	
condition	string	0	
action	string	0	
flags	string	0	
commname	string		The name to use on the commlink
commtype	string		the type of comms to construct
commdest, destination	string		the default communication destination as a string
continuous	flag	0	
sampled	flag	0	
comm_enabled	flag	0	
comms	flag	0	
usecomms	flag	0	
resettable	flag	0	
powerflow_check	flag	0	

Table A81: Set options for gridRelay objects. See Table A3 for additional settable options through gridPrimary

string(s)	type	default	description
reclosetime, reclosetime1	number	1	first reclose time
reclosetime2	number	5	second reclose time
maxrecloseattempts	number	0	total number of recloses
reclosers	number	0	
minclearingtime, cleartime	number	0	minimum clearing time for from bus breaker
limit	number	1	maximum current in puA
reclosertap, tap	number	0	From side tap multiplier
terminal	number	0	
recloserresettime, resettime	number	60	time the breaker has to be on before the recloser count resets
nondirectional	flag	0	

Table A82: Set options for breaker objects. See Table A81 for additional settable options through gridRelay

string(s)	type	default	description
cutoutvoltage, voltagelimit	number	0	
cutoutfrequency, freqlimit	number	0	
delay, voltagedelay	number	0	
frequencydelay	number	0	

Table A83: Set options for busRelay objects. See Table A81 for additional settable options through gridRelay

string(s)	type	default	description
autoname	number	-1	
delay	number	0	
terminal	number	0	
noreply	flag	0	

Table A84: Set options for controlRelay objects. See Table A81 for additional settable options through gridRelay

string(s)	type	default	description
delay	number	0.08	the delay time from first onset to trigger action
level, max_difference	number	0.2	the maximum allowable differential
reset_margin	number	0.01	the reset margin for clearing a fault
minlevel	number	0.01	the minimum absolute level to trigger for relative differential mode
relative	flag	0	
absolute	flag	0	

Table A85: Set options for differentialRelay objects. See Table A81 for additional settable options through gridRelay

string(s)	type	default	description
limit	number	1e+48	maximum current in puA
i2t	number	0	I squared t characteristic of fuse 1 in puA$\hat{2}$*s
terminal	number	0	
minblowtime	number	0.001	the minimum time required to for the fuse to blow only used if I2T>0;

Table A86: Set options for fuse objects. See Table A81 for additional settable options through gridRelay

string(s)	type	default	description
cutoutvoltage, voltagelimit	number	0	
cutoutfrequency, freqlimit	number	0	
delay, voltagedelay	number	0	
frequencydelay	number	0	
offtime	number	1e+48	
nondirectional	flag	0	

Table A87: Set options for loadRelay objects. See Table A81 for additional settable options through gridRelay

string(s)	type	default	description
terminal	number	0	
direct	number	0	
output, outputs	number	0	
input	string	0	
condition	string	0	
filter	string	0	
outputname, outputnames	string	0	
output, outputs	string	0	
process	string	0	
direct_io	flag	0	
direct	flag	0	

Table A88: Set options for sensor objects. See Table A81 for additional settable options through gridRelay

string(s)	type	default	description
zones	number	0	
terminal	number	0	
side	number	0	
resetmargin, margin	number	0.01	! the reset margin for clearing a fault
autoname	number	-1	storage for indicator of the type of autoname to use
levels	string	0	
delay	string	0	
nondirectional	flag	0	

Table A89: Set options for zonalRelay objects. See Table A81 for additional settable options through gridRelay

string(s)	type	default	description
deadband	number	20	
beta	number	8	
ki	number	0.005	
kp	number	1	
tf	number	8	
tr	number	15	

Table A90: Set options for AGControl objects. See Table A5 for additional settable options through gridSubModel

string(s)	type	default	description

Table A91: Set options for AGControlBattery objects. See Table A90 for additional settable options through AGControl

string(s)	type	default	description

Table A92: Set options for controlSystem objects. See Table A5 for additional settable options through gridSubModel

string(s)	type	default	description

Table A93: Set options for dispatcher objects. See Table A1 for additional settable options through gridCoreObject

string(s)	type	default	description
threshold, thresholdstart	number	1e+48	
thresholdstop	number	1e+48	
dispatchinterval, interval	number	300	

Table A94: Set options for reserveDispatcher objects. See Table A1 for additional settable options through gridCoreObject

string(s)	type	default	description
min	number	-1e+48	minimum set power
max	number	1e+48	maximum set power
target	number	0	
commtype	string		communication link type

Table A95: Set options for scheduler objects. See Table A5 for additional settable options through gridSubModel

string(s)	type	default	description
ramp, ramp10, ramp30, reserve	number	0	
rampup	number	1e+48	maximum ramp rate in the up direction
rampdown	number	1e+48	maximum ramp rate in the down direction
ramp10up	number	1e+48	The 10 minute maximum up ramp
ramp10down	number	1e+48	the 10 minute maximum down ramp
ramp30up	number	1e+48	the 30 minute maximum up ramp
ramp30down	number	1e+48	the 30 minute maximum down ramp
ramptime	number	1200	the ramp window
target	number	0	
reserveramptime	number	900	the time window the object has to meet the reserve
rampmode	string	0	

Table A96: Set options for schedulerRamp objects. See Table A95 for additional settable options through scheduler

string(s)	type	default	description
max	number		the maximum regulation an object has
min	number		the minimum regulation level
rampup, ramp	number		the rate at which the regulation must ramp
rampdown	number		the maximum rate at which regulation can ramp down
rating, base	number	100	generator base power
regfrac	number	0	
regupfrac	number	0	the capacity of the object to keep for regulation up purposes
regdownfrac	number	0	the capacity of the object to keep for regulation down purposes
regenabled	number	0	

Table A97: Set options for schedulerReg objects. See Table A96 for additional settable options through schedulerRamp

string(s)	type	default	description
initial, initiallife	number	150000	initial life in hours
basetemp	number	110	the temperature base for the lifespan equations
agingrate, agingconstant	number	14594	aging constant default value of 14594 based on research 15000 is another commonly used value
input	string	0	
input0	string	0	
useiec	flag	0	
iec	flag	0	
useieee	flag	0	
ieee	flag	0	
no_discconect	flag	0	

Table A98: Set options for txLifeSpan objects. See Table A88 for additional settable options through sensor

string(s)	type	default	description
ambient, ambienttemp	number	20	ambient temperature in C
dtempdt, temp_rate_of_change	number	0	rate of change of ambient temperature
dths, rated_hot_spot_rise, dthsr	number	35	hot spot rise temp at rated current over top oil
dttor, rated_top_oil_rise, dtto	number	45	Oil rise temp at rated current
ttor, oil_time_constant	number	4500	oil rise time constant
tgr, winding_time_constant	number	300	winding time constant
alarmtemp, alarmtemp1	number	0	the lower alarm temp
alarmtemp2	number	0	the level 2 alarm temp
cutouttemp	number	0	the temp at which the breakers are tripped
alarmdelay	number	300	delay time on the alarms and cutout;
lr, loss_ratio	number	6.5	Loss Ratio
m, winding_exponent, n, oil_exponent	number	1	winding exponent
txtype	string	0	
cooling	string	0	
auto	flag	0	
parameter_updates	flag	0	
enable_alarms	flag	0	

Table A99: Set options for txThermalModel objects. See Table A88 for additional settable options through sensor