Probabilistic Resource Adequacy Suite (PRAS) v0.6 Model Documentation

Gord Stephen

National Renewable Energy Laboratory

NREL is a national laboratory of the U.S. Department of Energy
Office of Energy Efficiency & Renewable Energy
Operated by the Alliance for Sustainable Energy, LLC

This report is available at no cost from the National Renewable Energy Laboratory (NREL) at www.nrel.gov/publications.

Contract No. DE-AC36-08GO28308

Technical Report
NREL/TP-5C00-79698
May 2021

Probabilistic Resource Adequacy Suite (PRAS) v0.6 Model Documentation

Gord Stephen

National Renewable Energy Laboratory

Suggested Citation

Stephen, Gord. 2021. *Probabilistic Resource Adequacy Suite (PRAS) v0.6 Model Documentation*. Golden, CO: National Renewable Energy Laboratory. NREL/TP-5C00-79698. https://www.nrel.gov/docs/fy21osti/79698.pdf.

Technical Report
NREL/TP-5C00-79698
May 2021

National Renewable Energy Laboratory
15013 Denver West Parkway
Golden, CO 80401
303-275-3000 • www.nrel.gov

Acknowledgments

Special thanks to NREL's Trieu Mai, Paul Denholm, and Greg Brinkman for their support and feedback in preparing this work, and to Sarah Awara and Sinnott Murphy (NREL), as well as Armando Figueroa Acevedo (Black & Veatch), for reviewing drafts of this report. Additional thanks go to Jaquelin Cochran, Bethany Frew, Wesley Cole, and Aaron Bloom for their broad historical support in the development of the PRAS model.

Contents

List of Figures

List of Tables

1 Introduction

The Probabilistic Resource Adequacy Suite, or PRAS, is a software package for studying power system resource adequacy. It allows the user to simulate power system operations under a wide range of operating conditions in order to study the risk of failing to meet demand (due to a lack of supply or deliverability), and identify the time periods and regions in which that risk occurs.

1.1 Resource Adequacy Background

An electrical power system is considered resource adequate if it has procured sufficient resources (including supply, transmission, and responsive demand) such that it runs a sufficiently low risk of invoking emergency measures (such as involuntary load shedding) due to resource unavailability or deliverability constraints. Resource adequacy is a necessary (but not sufficient) condition for overall power system reliability, which considers a broader set of system constraints including operational flexibility and the stability of system voltages and frequency.

Probabilistic resource adequacy assessment is the process by which resource shortfall risk is quantified. It involves mapping quantified uncertainties in system operating conditions (primarily forced outages of generators and lines) into probability distributions for operating outcomes of interest by simulating system operations under different probabilistically weighted scenarios. The nature of those simulations varies between models and can range from simple snapshot comparisons of peak demand versus available supply, through to chronological simulations of system dispatch and power flow over the full operating horizon.

The resulting outcomes can then be used to calculate industry-standard probabilistic risk metrics:

Expected Unserved Energy (EUE) is the expected (average) total energy shortfall over the study period. It may be expressed in energy units (e.g., GWh per year) or normalized against the system's total energy demand and expressed as a fraction (normalized EUE, or NEUE, expressed as a percentage or in parts-per-million, ppm).

Loss-of-Load Expectation (LOLE) is the expected (average) count of periods experiencing shortfall over the study period. It is expressed in terms of event-periods (e.g., event-hours per year, event-days per year). When reported in terms of event-hours, LOLE is sometimes referred to as LOLH (loss-of-load hours).

While a system's shortfall risk can never be eliminated entirely, if these risk metrics are assessed to be lower than some predetermined threshold, the system is considered resource adequate.

It can sometimes also be useful to express the average and/or incremental contribution of a particular resource to overall system adequacy in terms of capacity. This quantity (either in units of power, or as a fraction of the unit's nameplate capacity) is known as the capacity credit (sometimes called capacity value) of the resource. While many different methods are used to estimate the capacity credit of a resource, the most rigorous approaches generally involve assessing the change in probabilistic system adequacy associated with adding or removing the resource from the system. As a result, capacity credit calculation is often closely associated with probabilistic resource adequacy assessment.

1.2 Basic PRAS Structure and Usage

As illustrated in Figure 1, PRAS maps a provided representation of a power system to a probabilistic description of operational outcomes of interest, using a particular choice of operations simulation. The input system representation is called a "system model", the choice of operational representation is referred to as a "simulation specification", and different types of operating outcomes of interest are described by "result specifications".

PRAS is written in the Julia programming language and is controlled through the use of Julia scripts. The three components of a PRAS resource adequacy assessment (a system model, a simulation specification, and result specifications) map directly to the Julia function arguments required to launch a PRAS run. A typical resource adequacy assessment with PRAS involves creating or loading a system model, then invoking PRAS' `assess` function to perform the analysis:

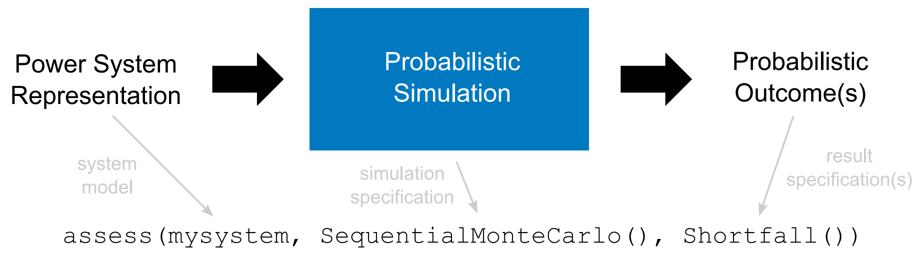

Figure 1. PRAS model structure and corresponding assessment function arguments

```
using PRAS

sys = SystemModel("filepath/to/mysystem.pras")

shortfallresult, flowresult =
    assess(sys, SequentialMonteCarlo(), Shortfall(), Flow())

eue, lole = EUE(shortfallresult), LOLE(shortfallresult)
```

More details on running PRAS via Julia code will be provided throughout the remainder of this report. In particular, Chapter 2 will detail the information contained in a system model input, Chapter 3 will explain the built-in simulation specification options in PRAS and the various modelling assumptions associated with each, and Chapter 4 will outline PRAS' available result specifications. Chapter 5 will cover how PRAS can calculate the capacity credit of specific resources based on the results of resource adequacy assessments, and Chapter 6 will provide information on extending PRAS beyond its built-in capabilities.

2 Input System Specification

Assessing the resource adequacy of a power system requires a description of the various resources available to that system, as well as its requirements for serving load. In PRAS, this involves representing the system's supply, storage, transmission, and demand characteristics in a specific data format. This information is stored in memory as a `SystemModel` Julia data structure, and on disk as an HDF5-formatted file with a `.pras` file extension. Loading the system data from disk to memory is accomplished via the following Julia code:

```
using PRAS
sys = SystemModel("filepath/to/mysystem.pras")
```

A full technical specification of the `.pras` storage format is available in the PRAS source code repository. Storing system data in this format ensures that it will remain readable in the future, even if PRAS' in-memory data representation changes. Newer versions of the PRAS package are always able to read `.pras` files created for older versions.

An in-memory `SystemModel` data structure can also be written back to disk:

```
savemodel(sys, "filepath/to/mynewsystem.pras")
```

PRAS simulates simplified power system operations over one or more consecutive time periods. The number of time periods to model and the temporal duration of a single time period are specified on a per-system basis, and must be consistent with provided starting and ending timestamps (defined with respect to a specific time zone).

When working with multiple years of weather data, a user may wish to create separate system models and perform runs for each year independently, or create a single system model containing the full multi-year dataset. The first approach can be useful for studying inter-annual variability of annual risk metrics using only the built-in methods – while this is also possible with a single multi-year run, it requires some additional post-processing work.

PRAS represents a power system as one or more **regions**, each containing zero or more **generators**, **storages**, and **generator-storages**. **Interfaces** contain **lines** and allow power transfer between two regions. Table 1 summarizes the characteristics of the different resource types (generators, storages, generator-storages, and lines), and the remainder of this section provides more details about each resource type and their associated resource collections (regions or interfaces).

2.1 Regions

PRAS does not represent the power system's individual electrical buses. Instead, PRAS **regions** are used to represent a collection of electrical buses that are grouped together for resource adequacy assessment purposes. Power transfer between buses within a single PRAS region is assumed to take place on a "copper sheet" with no intraregional transfer limits or line reliability limitations considered.

In a PRAS system representation, each region is associated with a descriptive name and an aggregate load time series, representing the total real power demand across all buses in the region, for every simulation period defined by the model.

2.2 Generators

Electrical supply resources with no modeled energy constraints (e.g., a thermal generator that can never exhaust its fuel supply) are represented in PRAS as **generators**. Generators are the simplest supply resource modeled in PRAS. In addition to a descriptive name and category, each generator unit is associated with a time series of maximum generating capacity. This time series can be a simple constant value (e.g., for a thermal plant) or can change in any arbitrary manner (e.g., for a solar PV array). Each generator is associated with a single PRAS region.

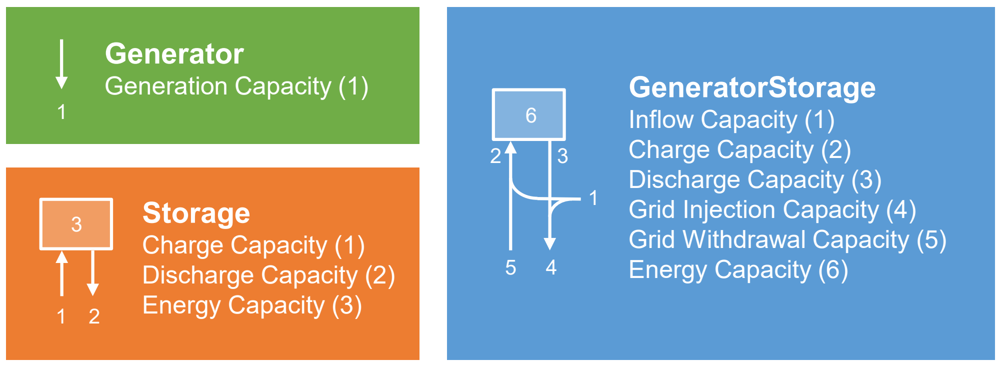

Figure 2. Relations between power and energy parameters for generator, storage, and generator-storage resources.

For each period of an operations simulation, each generator takes on one of two possible availability states. If the unit is available, it is capable of injecting power up to its maximum generation capacity (for that time period) in its associated region. If the unit is unavailable (representing some kind of unplanned or forced outage), it is incapable of injecting any power into the system. Between time periods, the unit may randomly transition to the opposite state according to unit-specific state transition probabilities. Like maximum available capacity, these transition probabilities are represented as time series, and so may be different during different time periods.

2.3 Storages

Resources that can shift electrical power availability forward in time but do not provide an overall net addition of energy into the system (e.g., a battery) are referred to as **storages** in PRAS. Like generators, storages are associated with descriptive name and category metadata. Each storage unit has both a charge and discharge capacity time series, representing the device's maximum ability to withdraw power from or inject power into the grid at a given point in time (as with generator capacity, these values may remain constant over the simulation or may vary to reflect external constraints).

Storage units also have a maximum energy capacity time series, reflecting the maximum amount of dischargeable energy the device can hold at a given point in time (increasing or decreasing this value will change the duration of time for which the device could charge or discharge at maximum power). The storage's state of charge increases with charging and decreases with discharging, and must always remain between zero and the maximum energy capacity in that time period. The energy flow relationships between these capacities are depicted visually in Figure 2.

If a storage device is charged and the maximum energy capacity decreases such that the state of charge exceeds the energy limit, the additional energy is automatically "spilled" (the surplus energy is not injected into the grid, but simply vanishes from the system).

Storage units may incur losses when moving energy into or out of the device (charge and discharge efficiency), or forward in time (carryover efficiency). When charging the unit, the effective increase to the state of charge is determined by multiplying the charging power by the charge efficiency. Similarly, when discharging the unit, the effective decrease to the state of charge is calculated by dividing the discharge power by the discharge efficiency. The available state of charge in the next time period is determined by multiplying the state of charge in the current time period by the carryover efficiency.

Just as with generators, storages may be in available or unavailable states, and move between these states randomly over time, according to provided state transition probabilities. Unavailable storages cannot inject power into or withdraw power from the grid, but they do maintain their energy state of charge during an outage (minus any carryover losses occurring over time).

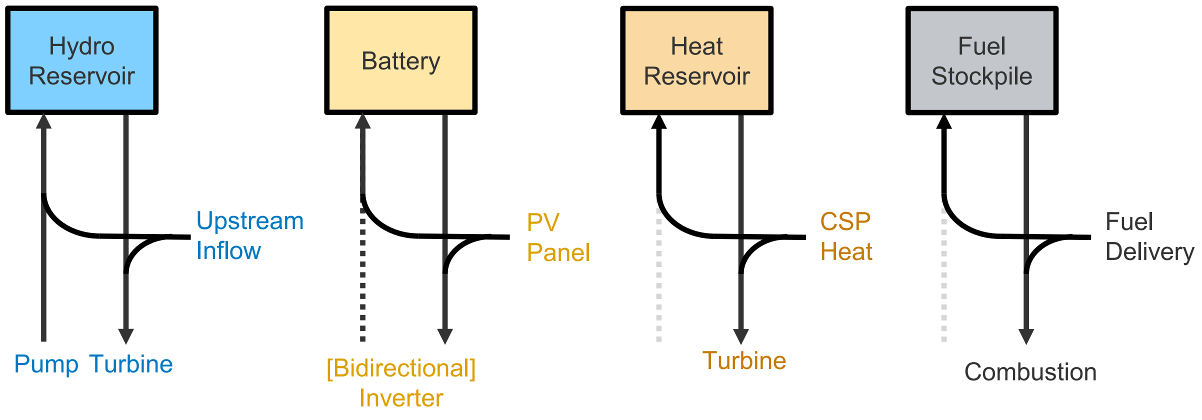

Figure 3. Example applications of the generator-storage resource type

2.4 Generator-Storages

Resources that add net new energy into the system but can also move that energy forward in time instead of injecting it immediately (see Figure 3 for examples) are referred to as **generator-storages** in PRAS. As the name suggests, they combine the characteristics of both generator and storage devices into a single unit.

As with generator and storage units, generator-storages have associated name and category metadata, and two availability states with random transition probabilities. They have a potentially time-varying maximum inflow capacity (representing potential new energy being added to the system and analogous to the generator's maximum generating capacity) as well as all the power and energy capacity and efficiency parameters associated with storages. They also have separate maximum grid injection and withdrawal capacity time series, reflecting the fact that (for example) they may not be able to discharge their internal storage at full capacity while simultaneously injecting their full exogenous energy inflow to the grid. The energy flow relationships between these capacities are depicted visually in Figure 2.

A generator-storage in the unavailable state can neither charge nor discharge its storage, nor send energy inflow to the grid. Like storage, it does retain its state of charge during outages (subject to carryover losses).

2.5 Interfaces

Interfaces define a potential capability to directly exchange power between two regions. Any set of two regions can have at most one interface connecting them. Each interface has both a "forward" and "backward" time-varying maximum transfer capability: the maximum "forward" transfer capability refers to the largest amount of total net power that can be moved from the first region to the second at a given point of time. Similarly, the maximum "backward" transfer capability refers to the largest amount of total net power that can be moved from the second region to the first.

2.6 Lines

Individual **lines** are assigned to a single specific interface and enable moving power between the two regions defined by that interface. Like other resources, a line is associated with name and category metadata, and transitions randomly between two availability states according to potentially time-varying transition probabilities. Like interfaces, lines have a potentially time-varying "forward" and "backward" transfer capability, where the forward and backward directions match those associated with the line's interface.

The total interregional transfer capability of an interface in a given direction is the lower of either the sum of transfer limits of available lines in that interface, or the interface-level transfer limit. A line in the unavailable state cannot move power between regions, and so does not contribute to the corresponding interface's sum of line-level transfer limits.

Table 1. PRAS resource parameters. Parameters in *italic* are fixed values: all others are provided as a time series.

Parameter	Generator	Storage	Generator-Storage	Line
Associated with a(n)...	*Region*	*Region*	*Region*	*Interface*
Name	●	●	●	●
Category	●	●	●	●
Generation Capacity	●			
Inflow Capacity			●	
Charge Capacity		●	●	
Discharge Capacity		●	●	
Energy Capacity		●	●	
Charge Efficiency		●	●	
Discharge Efficiency		●	●	
Carryover Efficiency		●	●	
Grid Injection Capacity			●	
Grid Withdrawal Capacity			●	
Forward Transfer Capacity				●
Backward Transfer Capacity				●
Available→Unavailable Transition Probability	●	●	●	●
Unavailable→Available Transition Probability	●	●	●	●

3 Simulation Specifications

There are many different simplifying assumptions that can be made when simulating power system operations for the purpose of studying resource adequacy. The level of simplification a modeler is willing to accept will depend on the goals of the study and the computational resources available to carry out the modeling exercise.

PRAS is referred to as a "suite" because of its inclusion of multiple power system operations models of varying fidelity and computational complexity. Each PRAS analysis (a single invocation of PRAS' `assess` function) is associated with exactly one of these operational models, or "simulation specifications". A simulation specification encodes a particular set of assumptions and simplifications that will be used when simulating operations in order to assess the resource adequacy of the study system.

The current version of PRAS includes three simulation specifications, **Convolution**, **Non-Sequential Monte Carlo**, and **Sequential Monte Carlo**, with additional user-defined specifications possible (see Section 6.1). The remainder of this section describes the methods and underlying assumptions of each of these built-in simulation specifications.

3.1 Convolution

The Convolution simulation specification is the simplest operational model in PRAS, making it the fastest to run but also the lowest fidelity. It corresponds to a classical Capacity Outage Probability Table (COPT) analysis (Billinton 1970), where an average forced outage rate (FOR) for each generator is used to determine the long-run probability distribution of total system-wide available (or unavailable) generating capacity in each chronological period, assuming independent generator failures. Exact system risk metrics can then be calculated from the combination of system load and this distribution.

3.1.1 Theory and Assumptions

The total available capacity distribution is calculated by adding together the random variables (convolving together the distributions) associated with available capacity for each unit in the system. The resulting probability distribution can change in each time period according to the provided availability state transition probabilities, which determine the forced outage rate as follows:

$$\text{FOR} = \frac{P(\text{Available} \rightarrow \text{Unavailable})}{P(\text{Available} \rightarrow \text{Unavailable}) + P(\text{Unavailable} \rightarrow \text{Available})}$$

Once probability distributions for total available capacity have been calculated for each time period, exact system-level risk metrics (LOLP and EUE) for each time period can be directly computed, which are then used to calculate risk metrics (LOLE and EUE) for the overall simulation horizon.

This method only considers generation adequacy, and assumes a "copper sheet" transmission infrastructure where all available generation capacity is deliverable to all load. The locations, transfer limits, and reliability parameters of individual transmission lines are ignored.

This method also ignores the intertemporal nature of power system operations, assessing adequacy in each time period independently of all others. The operating constraints of energy limited resources cannot be properly represented, and so such resources (storages and generator-storages) are ignored when performing simulations using this model of operations.

3.1.2 Usage

A convolution-based resource adequacy assessment is invoked by calling PRAS' `assess` method in Julia, with `Convolution()` as the simulation specification argument:

```
assess(sys, Convolution(), Shortfall())
```

The `Convolution()` specification accepts multiple optional keyword arguments, which can be provided in any order:

threaded A boolean value defaulting to `true`. If `true`, PRAS will parallelize probability distribution calculation across the number of threads available to Julia. Setting this to `false` can help with debugging if an assessment is hanging.

verbose A boolean value defaulting to `false`. If `true`, PRAS will output informative text describing the progress of the assessment.

3.2 Non-Sequential Monte Carlo

The Non-Sequential Monte Carlo method extends the Convolution method with the ability to model inter-regional power transfer limits and line outages (energy-limited resources are still ignored). It represents a compromise in fidelity and speed between the simpler Convolution method and a more computationally intensive Sequential Monte Carlo simulation.

3.2.1 Theory and Assumptions

Whereas the Convolution method combines all generators into a single COPT / available capacity distribution for the entire system (per time period), the Non-Sequential Monte Carlo method generates one distribution for each region (derived from the generators in that region) and one distribution for each interregional transmission interface (derived from lines in that interface), per time period.

Since generators in different regions are no longer collapsed into a single probability distribution, the number of unique system states is generally too large to allow enumerating through all of them. Instead, system states for each time period are representatively sampled by drawing total available generating capacities for each region and total available transfer capacities for each interface. These random samples are then used to estimate shortfall risk metrics using Monte Carlo sampling.

Each set of sampled parameters is used to formulate the “pipe-and-bubble” network flow problem shown in Figure 4, in which local demand in each region is satisfied (or not) using a combination of local generation, imported power, and unserved energy. Regions with surplus capacity can export that power to their neighbors if transmission limits allow, with a small transfer penalty applied to prevent loop flows. Any demand that cannot be supplied under the randomly drawn generation and transmission limits is considered unserved.

A predetermined number of operating condition samples are drawn for each time period in order to estimate shortfall risk metrics (LOLP and EUE) for that period. Like in the Convolution method, these independent period-level metrics are then combined to calculate metrics (LOLE and EUE) for the entire simulation horizon.

3.2.2 Usage

A Non-Sequential Monte Carlo resource adequacy assessment is invoked by calling PRAS’ `assess` method in Julia, with `NonSequentialMonteCarlo()` as the simulation specification argument:

```
assess(sys, NonSequentialMonteCarlo(), Shortfall())
```

The `NonSequentialMonteCarlo()` specification accepts several optional keyword arguments, which can be provided in any order:

samples A positive integer value defaulting to `10000`. It defines the number of samples (replications) to be used in the Monte Carlo simulation process.

seed An integer defaulting to a random value. It defines the seed to be used for random number generation when sampling generator and line availability.

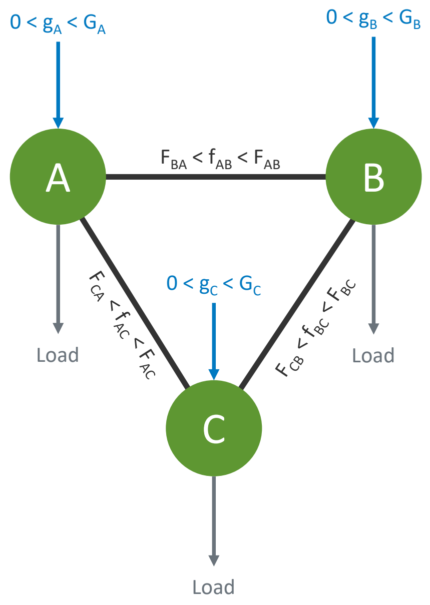
$0 < g_A < G_A$
$0 < g_B < G_B$
$F_{BA} < f_{AB} < F_{AB}$
A
B
$F_{CA} < f_{AC} < F_{AC}$
$0 < g_C < G_C$
$F_{CB} < f_{BC} < F_{BC}$
Load
Load
C
Load

Figure 4. Example three-region network flow problem to be solved by the Non-Sequential Monte Carlo simulation specification. Any load that cannot be served within the sampled generation and transmission constraints goes unserved.

threaded A boolean value defaulting to `true`. If `true`, PRAS will parallelize simulations across the number of threads available to Julia. Setting this to `false` can help with debugging if an assessment is hanging.

verbose A boolean value defaulting to `false`. If `true`, PRAS will output informative text describing the progress of the assessment.

3.3 Sequential Monte Carlo

The Sequential Monte Carlo method is the most detailed simulation specification included in PRAS. Unlike the Convolution and Non-Sequential Monte Carlo methods, it simulates the chronological evolution of the system, tracking individual unit-level outage states and the state of charge of energy-limited resources. While it is the most computationally intensive simulation method provided in PRAS, it remains much simpler (and therefore runs much faster) than a production cost model.

3.3.1 *Theory and Assumptions*

While the Convolution and Non-Sequential Monte Carlo methods sample a system or region's total available capacity from a probability distribution, the Sequential Monte Carlo method simulates unit-level outages using a two-state Markov model. In each time period, the availability state of each generator, storage, generator-storage, and line either changes or remains the same, at random, based on the unit's provided state transition probabilities. The capacities from each available generator (or line) in a given time period are then added together to determine the total available generating (transfer) capacity for a region (interface). Storage and generator-storage units are similarly enabled or disabled based on their availability states.

Like with the Non-Sequential Monte Carlo simulation specification, pipe-and-bubble power transfers between regions are possible, subject to interface and line transfer limits, and there is a small penalty applied to transfers in order to prevent loop flows.

The Sequential Monte Carlo method is unique in its ability to represent energy-limited resources (storages and generator-storages). These resources are dispatched conservatively so as to approximately minimize unserved energy over the full simulation horizon, charging from the grid whenever surplus generating capacity is available, and discharging only when needed to avoid or mitigate unserved energy. Charging and discharging is coordinated between resources using the time-to-go priority described in Evans, Tindemans, and Angeli (2019): resources that would be able to discharge the longest at their maximum rate are discharged first, and resources that would take the longest time to charge at their maximum charge rate are charged first. Cross-charging (discharging one resource in order to charge another) is not permitted.

In Sequential Monte Carlo simulations, one "sample" involves chronological simulation of the system over the full operating horizon. Unserved energy results for each hour and the overall horizon are recorded before restarting the simulation and repeating the process with new random outage draws. Once all samples have been completed, hourly and overall system risk metrics can be calculated.

3.3.2 Usage

A sequential Monte Carlo resource adequacy assessment is invoked by calling PRAS' `assess` method in Julia, with `SequentialMonteCarlo()` as the simulation specification argument:

```
assess(sys, SequentialMonteCarlo(), Shortfall())
```

The `SequentialMonteCarlo()` specification accepts several optional keyword arguments, which can be provided in any order:

samples A positive integer value defaulting to `10000`. It defines the number of samples (replications) to be used in the Monte Carlo simulation process.

seed An integer defaulting to a random value. It defines the seed to be used for random number generation when sampling generator and line outage state transitions.

threaded A boolean value defaulting to `true`. If `true`, PRAS will parallelize simulations across the number of threads available to Julia. Setting this to `false` can help with debugging if an assessment is hanging.

verbose A boolean value defaulting to `false`. If `true`, PRAS will output informative text describing the progress of the assessment.

4 Result Specifications

Different analyses require different kinds of results, and different levels of detail within those results. PRAS considers many operational decisions and system states internally, not all of which are relevant outputs for every analysis. When a user invokes PRAS' `assess` function, one or more "result specifications" must be provided in order to indicate the simulation outcomes that are of interest, and the desired level of sample aggregation or unit type (if applicable) for which those results should be reported. In general, sample-level disaggregation should be used with care, as this can require large amounts of memory if simulating with many samples.

The current version of PRAS includes six built-in result specification families, with additional user-defined specifications possible (see Section 6.2). These families can be classified into regional results (**Shortfall** and **Surplus**), interface results (**Flow** and **Utilization**), and unit results (**Availability** and **Energy**).

Not all result specification families or disaggregation variants within a family are available for use with every simulation specification. For example, the Convolution simulation specification is an exact analytical method and does not model transmission or consider unit-level availability, so neither interface results, unit results, nor sample-level result disaggregations are available.

When invoking `assess` in Julia, result specifications are provided as the final arguments to the function call, and a tuple of results are returned in that same order. (Note that a tuple is *always* returned, even if a single result specification is requested.) An example of requesting three result specifications is:

```
surplus, flow, genavail = assess(
    sys, SequentialMonteCarlo(), Surplus(), Flow(), GeneratorAvailability())
```

Depending on the result specification, a result object may support indexing into it to obtain results for a specific time period, region, interface, or unit. For example, using the results returned above:

```
timestamp = ZonedDateTime(2020, 1, 1, 13, tz"UTC")
genname = "Generator 1"
regionname = "Region A"
interface = "Region A" => "Region B"

# get the sample mean and standard deviation of observed total system
# surplus capacity at 1pm UTC on January 1, 2020:
m, sd = surplus[timestamp]

# get the sample mean and standard deviation of observed surplus capacity
# in Region A at 1pm UTC on January 1, 2020:
m, sd = surplus[regionname, timestamp]

# get the sample mean and standard deviation of average interface flow
# between Region A and Region B:
m, sd = flow[interface]

# get the sample mean and standard deviation of interface flow
# between Region A and Region B at 1pm UTC on January 1, 2020:
m, sd = flow[interface, timestamp]

# get the vector of random generator availability states in every sample
# for Generator 1, at 1pm UTC on January 1, 2020:
states = genavail[genname, timestamp]
```

Results can be reported in different ways depending on the result specification being used, and not all types of indexing are appropriate for every result specification. For example, it would not make sense to aggregate interface flows across all interfaces in the system, or surplus power (potentially from energy-limited devices) across all time periods.

The remainder of this chapter provides additional details about the six built-in result specification families in PRAS.

4.1 Regional Results

The Shortfall and Surplus result families are defined over regions, and their result objects can all be indexed into by region name. Table 2 outlines the simulation specifications that members of these families are compatible with, as well as the levels of disaggregation they support.

Table 2. Regional result specification characteristics.
(Conv = Convolution, NSMC = Non-Sequential Monte Carlo, SMC = Sequential Monte Carlo)

Result Specification	Units	*supported by...*			*get results by...*			
		Conv	NSMC	SMC	Sample	Region	Timestep	Region + Timestep
`Shortfall`	Energy	●	●	●		●	●	●
`ShortfallSamples`	Energy		●	●	●	●	●	●
`Surplus`	Power	●	●	●			●	●
`SurplusSamples`	Power		●	●	●		●	●

4.1.1 Shortfall

The Shortfall family of result specifications (`Shortfall` and `ShortfallSamples`) reports on unserved energy occurring during simulations. As quantifying unserved energy is the core aspect of resource adequacy analysis, in practice almost every assessment requests a Shortfall-related result. The basic `Shortfall` specification is most commonly used and reports average shortfall results, while `ShortfallSamples` provides more detailed results at the level of individual simulations (samples).

Shortfall result objects can be indexed into by region, timestep, both region and timestep, or neither. Indexing on neither (via `result[]`) reports the total shortfall across all regions and time periods.

Shortfall results are unique among the built-in result types in that the raw results can also be converted to specific probabilistic risk metrics (**EUE** and **LOLE**). For sampling-based methods, both metric estimates and the standard error of those estimates are provided. For example, after assessing the system, metrics across all regions and the full simulation horizon can be extracted as:

```
shortfall, = assess(sys, SequentialMonteCarlo(), Shortfall())
eue_overall = EUE(shortfall)
lole_overall = LOLE(shortfall)
```

More specific metrics can be obtained as well:

```
region = "Region A"
period = ZonedDateTime(2020, 1, 1, 0, tz"America/Denver")

eue_period = EUE(shortfall, period)
lole_region = LOLE(shortfall, region)
eue_region_period = EUE(shortfall, region, period)
```

4.1.2 Surplus

The Surplus family of result specifications (`Surplus` and `SurplusSamples`) reports on excess grid injection capacity (via generation or discharging) in the system. This can be used to study "near misses" where shortfall came close to occurring but did not actually happen. The `Surplus` specification reports average surplus across samples, while `SurplusSamples` reports simulation-level observations.

Surplus capacity is reported in terms of power, and so results are always disaggregated by timestep (indexed either by timestep or both region and timestep).

4.2 Interface Results

The Flow and Utilization families of result specifications are defined over interfaces, and their result objects can all be indexed into by a pair of region names (indicating the source and destination regions for power transfer). Table 3 outlines the simulation specifications that members of these families are compatible with, as well as the levels of disaggregation they support.

Table 3. Interface result specification characteristics.
(Conv = Convolution, NSMC = Non-Sequential Monte Carlo, SMC = Sequential Monte Carlo)

		supported by...			*get results by...*			
Result Specification	Units	Conv	NSMC	SMC	Sample	Interface	Timestep	Interface + Timestep
`Flow`	Power		●	●		●		●
`FlowSamples`			●	●	●	●		●
`Utilization`	–		●	●		●		●
`UtilizationSamples`			●	●	●	●		●

4.2.1 Flow

The Flow family of result specifications (`Flow` and `FlowSamples`) reports the direction and magnitude of power transfer on an interface. This can be used to study which regions are importers vs exporters of energy, either on average or at specific periods in time. The `Flow` specification reports average flow across all samples, while `FlowSamples` reports simulation-level observations. Flow results are directional, so the order in which the regions are provided when looking up a result will determine the result's sign. For example:

```
m1, sd1 = flow["Region A" => "Region B"]
m2, sd2 = flow["Region B" => "Region A"]

m1 == -m2 # true
sd1 == sd2 # true
```

Flow values are reported in terms of power, and results are always disaggregated by interface. Results that aggregate over time report the average flow over the time span.

4.2.2 Utilization

The Utilization family of result specifications (`Utilization` and `UtilizationSamples`) is similar to the Flow family, but reports the fraction of an interface's available transfer capacity that is used in the direction of flow, instead of the flow power itself. Results can therefore range between 0 and 1. This metric can be useful for studying the impact of line outages and transmission congestion on unserved energy. The `Utilization` specification reports average flow across all samples, while `UtilizationSamples` reports simulation-level observations. Unlike Flow, Utilization results are not directional and so will report the same utilization regardless of the flow direction implied by the order of the provided regions:

```
util, = assess(sys, SequentialMonteCarlo(), Utilization())
util["Region A" => "Region B"] == util["Region B" => "Region A"]
```

Utilization values are unitless, and results are always disaggregated by interface. Results that aggregate over time report the average utilization over the time span.

4.3 Unit Results

The Availability and Energy families of result specifications are defined over individual units, and their result objects can all be indexed into by a unit name and timestep. Table 4 outlines the simulation specifications that members of these families are compatible with, as well as the levels of disaggregation they support.

Table 4. Unit result specification characteristics.
(Conv = Convolution, NSMC = Non-Sequential Monte Carlo, SMC = Sequential Monte Carlo)

Result Specification	Units	*supported by...*			*get results by...*			
		Conv	NSMC	SMC	Sample	Unit	Timestep	Unit + Timestep
`GeneratorAvailability`	–			●	●			●
`StorageAvailability`				●	●			●
`GeneratorStorageAvailability`				●	●			●
`LineAvailability`				●	●			●
`StorageEnergy`	Energy			●			●	●
`StorageEnergySamples`				●	●		●	●
`GeneratorStorageEnergy`				●			●	●
`GeneratorStorageEnergySamples`				●	●		●	●

4.3.1 Availability

The Availability family of result specifications (`GeneratorAvailability`, `StorageAvailability`, `GeneratorStorageAvailability`, and `LineAvailability`) reports the availability state (available, or unavailable due to an unplanned outage) of individual units in the simulation. The four result specification variants correspond to the four categories of resources: generators, storages, generator-storages, and lines. Availability is reported as a boolean value (with `true` indicating the unit is available, and `false` indicating it isn't), and is always disaggregated by unit, timestep, and sample.

4.3.2 Energy

The Energy family of result specifications (`StorageEnergy`, `StorageEnergySamples`, `GeneratorStorageEnergy`, and `GeneratorStorageEnergySamples`) reports the energy state-of-charge associated with individual energy-limited resources. Result specification variants are available for selecting the category of energy-limited resource (storage or generator-storage) to report, as well as for requesting sample-level disaggregation. Energy is always disaggregated by timestep and may also be disaggregated by unit (get the state of charge of a single unit) or aggregated across the system (get the sum of states of charge of all storage devices in the system).

4.4 Additional Examples

This section provides a more complete example of running a PRAS assessment, with a hypothetical analysis process making use of multiple different results.

```
using PRAS
```

```
# Load in a system model from a .pras file.
# This hypothetical system has an hourly time resolution with an
# extent / simulation horizon of one year.
sys = SystemModel("mysystem.pras")

# This system has multiple regions and relies on battery storage, so
# run a sequential Monte Carlo analysis:
shortfall, utilization, storage = assess(
    sys, SequentialMonteCarlo(samples=10000, seed=1),
    Shortfall(), Utilization(), StorageEnergy())

# Start by checking the overall system adequacy:
lole = LOLE(shortfall) # event-hours per year
eue = EUE(shortfall) # unserved energy per year
```

Suppose LOLE is below the target threshold but EUE seems high, suggesting large amounts of unserved energy are concentrated in a small number of hours. What do the hourly results show?

```
# Note 1: LOLE.(shortfall, many_hours) is Julia shorthand for calling LOLE
#         on every timestep in the collection many_hours
# Note 2: Here results are in terms of event-hours per hour, which is
#         equivalent to the loss-of-load probability (LOLP) for each hour
lolps = LOLE.(shortfall, sys.timestamps)
```

One might see that a particular hour has an LOLP near 1.0, indicating that load is consistently getting dropped in that period. Is this a local issue or system-wide? One can check the unserved energy by region in that hour:

```
shortfall_period = ZonedDateTime(2020, 8, 21, 17, tz"America/Denver")
unserved_by_region = EUE.(shortfall, sys.regions.names, shortfall_period)
```

Perhaps only one region (D) has non-zero EUE in that hour, indicating that this must be a load pocket issue. We can also look at the utilization of interfaces into that region in that period:

```
utilization["Region A" => "Region D", shortfall_period]
utilization["Region B" => "Region D", shortfall_period]
utilization["Region C" => "Region D", shortfall_period]
```

These sample-averaged utilizations should all be very close to 1.0, indicating that power transfers are consistently maxed out; neighboring regions have power available but can't send it to Region D.

Transmission expansion is clearly one solution to this adequacy issue. Is local storage another alternative? One can check on the average state-of-charge of the existing battery in that region, both in the hour before and during the problematic period:

```
storage["Battery D1", shortfall_period-Hour(1)]
storage["Battery D1", shortfall_period]
```

It may be that the battery is on average fully charged going into the event, and perhaps retains some energy during the event, even as load is being dropped. The device's ability to mitigate the shortfall must then be limited only by its discharge capacity, so given that the event doesn't last long, adding additional short-duration storage in this region would help.

Note that if the event occurred less consistently, this analysis could also have been performed on the subset of samples in which the event was observed, using the `ShortfallSamples`, `UtilizationSamples`, and `StorageEnergySamples` result specifications instead.

5 Capacity Credit Calculation

Resource adequacy paradigms premised on adding resource capacities together to meet a planning reserve margin require the quantification of a "capacity credit" (sometimes called "capacity value") for individual resources. While the process of assigning capacity credits is relatively straightforward for thermal generating units with consistent potential contributions to system adequacy throughout the day and year (assuming no fuel constraints), the contributions of variable and energy-limited resources can be much more difficult to represent as a single capacity rating.

In these cases, an accurate characterization depends on the broader system context in which the resource operates. Probabilistically derived capacity credit calculations provide a technology-agnostic means of expressing the contributions of different resources (with diverse and potentially complicated operating characteristics and constraints) in terms of a common, simple measure of capacity.

PRAS provides two different methods for mapping incremental resource adequacy contributions to generic capacity: Equivalent Firm Capacity (**EFC**) and Effective Load Carrying Capability (**ELCC**). In each case, the user must provide PRAS with two system representations: one that contains the study resource (the augmented system), and one that does not (the base system). The difference between the two systems is then quantified in terms of a capacity credit.

By choosing what is included in the base case relative to the augmented case, the user can study either the average, portfolio-level capacity credit of a resource class (by excluding all resources of that class from the base case, and including them all in the augmented case) or the marginal capacity credit (by including almost all of the resource type in the base case, and adding a single incremental unit in the augmented case).

Note that probabilistically derived capacity credit calculations always involve some kind of measurement of the reduction in system risk associated with moving from the base system to the augmented system. If the base system's risk cannot be reduced (perhaps because the base system's shortfall risk is too small to obtain a non-zero estimate, or because shortfall only occurs in load pockets elsewhere in the system), adequacy-based capacity credit metrics may not be meaningful. In these cases, the starting system may need to be modified, or a different capacity credit calculation method may be required.

The remainder of this chapter provides details on the theoretical and practical aspects of using EFC and ELCC for capacity credit analysis in PRAS. Further mathematical details regarding capacity credit are available in Zachary and Dent (2011).

5.1 Equivalent Firm Capacity (EFC)

5.1.1 Theory

EFC calculates the amount of idealized "firm" capacity (uniformly available across all periods, without ever going on outage) that is required to reproduce the observed resource adequacy benefit (reduction of a specific risk metric) associated with some study resource of interest. It requires both a base case system (without the study resource added) and an augmented system (with the study resource added). The analysis then proceeds as follows:

1. Assess the shortfall risk of the base system according to the chosen metric (EUE or LOLE).
2. Assess the (lower) shortfall risk of the augmented system according to the chosen metric.
3. Reassess the shortfall risk of the base system after adding some amount of "firm" capacity. If the risk matches that of the augmented system, stop. The amount of firm capacity added is the Equivalent Firm Capacity of the study resource.
4. If the base+firm and augmented system risks do not match, change the amount of firm capacity added to the base system, repeating until the chosen shortfall risk metrics for each system match.

Typically, the counterfactual firm capacity is added to the system as a direct replacement for the study resource, and so is located in the same region (or distributed across multiple regions in corresponding proportions) as the study resource. PRAS uses a bisection method to find the appropriate total firm capacity to add to the base system.

5.1.2 Usage

Performing an EFC assessment in PRAS requires specifying two different `SystemModels`: one representing the base system, and a second representing the augmented system. It also requires specifying the probabilistic risk metric to use when comparing system risks, an upper bound on the EFC (usually, the nameplate capacity of the study resource) and to which region(s) the counterfactual firm capacity should be added. Finally, the simulation specification should be provided (any simulation method can be used).

For example, to calculate EFC based on EUE for a resource in region A, with an upper EFC bound of 1000 MW (assuming the `SystemModels` are represented in MW), using the sequential Monte Carlo simulation specification:

```
assess(base_system, augmented_system,
       EFC{EUE}(1000, "A"), SequentialMonteCarlo())
```

If the study resources are spread over multiple regions (for example, 600 MW of wind in region A and 400 MW of wind in region B), the fraction of total firm capacity added to each region can be specified as:

```
assess(base_system, augmented_system,
       EFC{EUE}(1000, ["A"=>0.6, "B"=>0.4]), SequentialMonteCarlo())
```

The `EFC()` specification accepts multiple optional keyword arguments, which can be provided in any order:

p_value A floating point value giving the maximum allowable p-value from a one-sided hypothesis test. The test considers whether the lower risk metric used during bisection is in fact less than the upper risk metric. If the p-value exceeds this level, the assessment will terminate early due to a lack of statistical power. Note that this only matters for simulation specifications returning estimates with non-zero standard errors, i.e. Monte Carlo-based methods. Defaults to `0.05`.

capacity_gap An integer giving the maximum desired difference between reported upper and lower bounds on capacity credit. Once the gap between upper and lower bounds is less than or equal to this value, the assessment will terminate. Defaults to `1`.

verbose A boolean value defaulting to `false`. If `true`, PRAS will output informative text describing the progress of the assessment.

5.2 Effective Load Carrying Capability (ELCC)

5.2.1 Theory

ELCC quantifies the capacity credit of a study resource according to the amount of additional constant load the system can serve while maintaining the same shortfall risk. Like EFC, it requires both a base case system (without the study resource added) and an augmented system (with the study resource added). The analysis then proceeds as follows:

1. Assess the shortfall risk of the base system according to the chosen metric (EUE or LOLE).
2. Assess the (lower) shortfall risk of the augmented system according to the chosen metric.
3. Reassess the shortfall risk of the augmented system after adding some amount of constant load. If the risk matches that of the base system, stop. The amount of constant load added is the Effective Load Carrying Capability of the study resource.
4. If the base and augmented+load system risks do not match, change the amount of load added to the augmented system, repeating until the chosen shortfall risk metrics for each system match.

ELCC calculations in a multi-region system require choosing where load should be increased. There are many possible options, including uniformly distributing new load across each region, distributing load proportional to total energy demand in each region, and adding load only in the region with the study resource. The “correct” choice will depend on the goals of the specific analysis. Once the regional load distribution is specified, PRAS uses a bisection method to find the appropriate amount of total load to add to the system.

5.2.2 Usage

Performing an ELCC assessment in PRAS requires specifying two different `SystemModels`: one representing the base system, and a second representing the augmented system. It also requires specifying the probabilistic risk metric to use when comparing system risks, an upper bound on the ELCC (usually, the nameplate capacity of the study resource) and to which region(s) the additional load should be added. Finally, the simulation specification should be provided.

For example, to calculate ELCC based on EUE for a resource intending to serve load in region A, with an upper ELCC bound of 1000 MW (assuming the `SystemModels` are represented in MW), using the convolution simulation specification:

```
assess(base_system, augmented_system,
       ELCC{EUE}(1000, "A"), Convolution())
```

If the load serving assessment is to be spread over multiple regions (for example, 50% of load in region A and 50% in region B), the fraction of additional load added to each region can be specified as:

```
assess(base_system, augmented_system,
       ELCC{EUE}(1000, ["A"=>0.5, "B"=>0.5]), Convolution())
```

The `ELCC()` specification accepts multiple optional keyword arguments, which can be provided in any order:

p_value A floating point value giving the maximum allowable p-value from a one-sided hypothesis test. The test considers whether the lower risk metric used during bisection is in fact less than the upper risk metric. If the p-value exceeds this level, the assessment will terminate early due to a lack of statistical power. Note that this only matters for simulation specifications returning estimates with non-zero standard errors, i.e. Monte Carlo-based methods. Defaults to `0.05`.

capacity_gap An integer giving the maximum desired difference between reported upper and lower bounds on capacity credit. Once the gap between upper and lower bounds is less than or equal to this value, the assessment will terminate. Defaults to `1`.

verbose A boolean value defaulting to `false`. If `true`, PRAS will output informative text describing the progress of the assessment.

5.3 Capacity Credit Results

Both EFC and ELCC assessments return `CapacityCreditResult` objects. These objects contain information on estimated lower and upper bounds of the capacity credit, as well as additional details about the process through which the capacity credit was calculated. Results can be retrieved as follows:

```
cc_result = assess(base_system, augmented_system, EFC{EUE}(1000, "A"),
                   SequentialMonteCarlo())

# Get lower and upper bounds on CC estimate
cc_lower = minimum(cc_result)
cc_upper = maximum(cc_result)

# Get both bounds at once
cc_lower, cc_upper = extrema(cc_result)
```

6 Extending PRAS

PRAS provides opportunities for users to non-invasively build on its general simulation framework by redefining how simulations are executed, augmenting how results are reported, or both. This allows for customized analyses without requiring the user to modify code in the main PRAS package or implement their own model from scratch.

To implement custom functionality, a user needs to define specific Julia data structures as well as implement function methods that operate on those structures. Julia's multiple dispatch functionality can then identify and use these newly defined capabilities when the `assess` function is invoked appropriately.

6.1 Custom Simulation Specifications

Custom simulation specifications allow for redefining how PRAS models system operations. In addition to the data structures and methods listed here, defining a new simulation specification also requires defining the appropriate simulation-result interactions (see Section 6.3).

6.1.1 New Data Structure Requirements

The following new data structure (struct / type) should be defined in Julia:

Simulation Specification

The main type representing the new simulation specification. It should be a subtype of the `SimulationSpec` abstract type and can contain fields that store simulation parameters (such as the number of Monte Carlo samples to run or the random number generation seed to use). For example:

```
struct MyCustomSimSpec <: SimulationSpec
    nsamples::UInt64
    seed::UInt64
end
```

6.1.2 New Method Requirements

The following new function method should be defined in Julia:

assess

The method to be invoked when the `assess` function is called with the previously defined simulation specification. By convention, the method should take a `SystemModel` as the first argument, followed by a specific subtype of `SimulationSpec`, followed by one or more unspecified subtypes of `ResultSpec`. For example (using the `MyCustomSimSpec` type defined above):

```
function PRAS.assess(
    sys::SystemModel, simspec::MyCustomSimSpec, resultspecs::ResultSpec...)

    # Implement the simulation logic for MyCustomSimSpec here

    # This will include simulation-result interaction calls to result
    # recording methods, which will need to be implemented by any result
    # specification wanting to be compatible with MyCustomSimSpec

end
```

6.2 Custom Result Specifications

Custom result specifications allow for saving out additional information that may be generated during simulations of system operations. In addition to the data structures and methods listed here, defining a new result specification also requires defining the appropriate simulation-result interactions (see Section 6.3).

6.2.1 New Data Structure Requirements

The following new data structures (structs / types) should be defined in Julia:

Result Specification

The main type representing the result specification. It should be a subtype of the `ResultSpec` abstract type and can contain fields that store result parameters (although this is usually not necessary). For example:

```
struct MyCustomResultSpec <: ResultSpec
end
```

Result

The type of the data that is returned at the end of an assessment and stores any information to be reported to the end-user. It should be a subtype of the `Result` abstract type and should contain fields that store the desired results. For example:

```
struct MyCustomResult <: Result
    myoutput1::Float64
    myoutput2::Vector{Bool}
end
```

6.2.2 New Method Requirements

Indexing

Result data should support index lookups to report overall results or values for specific time periods, regions, interfaces, units, etc. The specifics of how the result data is indexed will depend on the nature of the result type, but will likely involve implementing one or more of the following methods (here we assume the new result type is `MyCustomResult`):

```
Base.getindex(result::MyCustomResult)
Base.getindex(result::MyCustomResult, region_or_unit::String)
Base.getindex(result::MyCustomResult, interface::Pair{String,String})
Base.getindex(result::MyCustomResult, period::ZonedDateTime)
Base.getindex(result::MyCustomResult,
              region_or_unit::String, period::ZonedDateTime)
Base.getindex(result::MyCustomResult,
              interface::Pair{String,String}, period::ZonedDateTime)
```

Risk Metrics

If the result includes information that can be used to calculate resource adequacy metrics, some or all of the following new function methods should be defined (here we assume the new result type is `MyCustomResult`):

```
PRAS.LOLE(result::MyCustomResult)
PRAS.LOLE(result::MyCustomResult, region::String)
PRAS.LOLE(result::MyCustomResult, period::ZonedDateTime)
PRAS.LOLE(result::MyCustomResult, region::String, period::ZonedDateTime)

PRAS.EUE(result::MyCustomResult)
PRAS.EUE(result::MyCustomResult, region::String)
PRAS.EUE(result::MyCustomResult, period::ZonedDateTime)
PRAS.EUE(result::MyCustomResult, region::String, period::ZonedDateTime)
```

If desired, new result specifications may define additional result-specific accessor methods as well.

6.3 Simulation-Result Interfaces

Result specifications need a way to map information produced by a simulation to outcomes of interest. The specifics of how this is implemented will vary between simulation specifications, but in general, a specific `assess` method will invoke another method that records abstract results. This recording method will then be implemented by all of the concrete result specifications wishing to support that simulation specification. A very simplified example of this pattern is:

```
function assess(
    sys::SystemModel, simspec::MyCustomSimSpec, resultspecs::ResultSpec...)

    # Implement the simulation logic for MyCustomSimSpec here,
    # and collect full results
    simulationdata = ...

    # Store requested results
    results = ()
    for resultspec in resultspecs
        results = (results..., record(simspec, resultspec, simulationdata))
    end

    return results

end

function record(
    simspec::MyCustomSimSpec, resultspec::Shortfall, simulationdata)
    # Map simulationdata to shortfall results here
    return ShortfallResult(...)
end

function record(
    simspec::MyCustomSimSpec, resultspec::Surplus, simulationdata)
    # Map simulationdata to surplus results here
    return SurplusResult(...)
end

function record(
    simspec::MyCustomSimSpec, resultspec::MyCustomResultSpec, simulationdata)
    # Map simulationdata to my custom results here
    return MyCustomResult(...)
end
```

The remainder of this section explains how this is accomplished for the `Convolution` and `SequentialMonteCarlo` result specifications in particular. By implementing the types and methods described here, a new result specification can be made compatible with these existing simulation types. In each case, we assume the `MyResultSpec <: ResultSpec` and `MyResult <: Result` types have been previously defined.

6.3.1 Convolution Interface

Result Accumulator

A convolution result accumulator incrementally collects relevant intermediate outcomes as available capacity distributions for different time periods are evaluated.

```
# Define the accumulator structure
struct ConvMyResultAccumulator <: ResultAccumulator{Convolution,MyResultSpec}
    # fields for holding intermediate data go here
end

# Help PRAS know which accumulator type to expect before one's created
PRAS.ResourceAdequacy.accumulatortype(::Convolution, ::MyResultSpec) =
    ConvMyResultAccumulator

# Initialize a new accumulator
function PRAS.ResourceAdequacy.accumulator(
    sys::SystemModel, simspec::Convolution, resultspec::MyResultSpec)
    return ConvMyResultAccumulator(...)
end
```

record!

Once unit outage rates and capacities in a given time period `t` have been processed (convolved) to create a probability distribution `distr` for shortfall or surplus capacity, the `record!` method extracts features of interest from the distribution and updates the corresponding accumulator `acc` in-place.

```
PRAS.ResourceAdequacy.record!(
    acc::ConvMyResultAccumulator, t::Int, distr::CapacityDistribution)
```

merge!

For multithreaded assessments, PRAS creates one accumulator per worker thread (parallel task) and merges each thread's accumulator information together once work is completed. `merge!` defines how an accumulator `a` should be updated in-place to incorporate the results obtained by another accumulator `b`.

```
PRAS.ResourceAdequacy.merge!(
    a::ConvMyResultAccumulator, b::ConvMyResultAccumulator)
```

finalize

Once all of the thread accumulators have been merged down to a single accumulator reflecting results from all of the threads, this final accumulator `acc` is mapped to the final result output through a `finalize` method.

```
function PRAS.ResourceAdequacy.finalize(
    acc::ConvMyResultAccumulator, sys::SystemModel)
    return MyResult(...)
end
```

6.3.2 Sequential Monte Carlo Interface

Result Accumulator

A sequential Monte Carlo result accumulator incrementally collects relevant intermediate outcomes as chronological simulations under different random samples are performed.

```
# Define the accumulator structure
struct SMCMyResultAccumulator <: ResultAccumulator{SequentialMonteCarlo,MyResultSpec}
    # fields for holding intermediate data go here
end

# Help PRAS know which accumulator type to expect before one's created
PRAS.ResourceAdequacy.accumulatortype(::SequentialMonteCarlo, ::MyResultSpec) =
    SMCMyResultAccumulator

# Initialize a new accumulator
function PRAS.ResourceAdequacy.accumulator(
    sys::SystemModel, simspec::SequentialMonteCarlo, resultspec::MyResultSpec)
    return SMCMyResultAccumulator(...)
end
```

record!

Once system operations in a given time period `t` have been simulated within a given chronological sample sequence `s`, the `record!` method extracts outcomes of interest from one or both of the system's current `state` and the solution to the period's dispatch problem `prob`. These results are used to update the accumulator `acc` in-place.

```
PRAS.ResourceAdequacy.record!(
    acc::SMCMyResultAccumulator, sys::SystemModel, state::SystemState,
    prob::DispatchProblem, s::Int, t::Int)
```

reset!

At the end of each chronological sequence of time periods `s`, the `reset!` method updates the accumulator `acc` in-place to finalize recording of any results requiring information from multiple periods and prepare the accumulator to start receiving values from a new chronological simulation sequence.

```
PRAS.ResourceAdequacy.reset!(acc::SMCMyResultAccumulator, s::Int)

# Often no action is required here,
# so a simple one-line implementation is possible
PRAS.ResourceAdequacy.reset!(acc::SMCMyResultAccumulator, s::Int) = nothing
```

merge!

Just as with the Convolution result spec, for multithreaded assessments PRAS creates one accumulator per worker thread (parallel task) and merges each thread's accumulator information together once work is completed. `merge!` defines how an accumulator `a` should be updated in-place to incorporate the results obtained by another accumulator `b`.

```
PRAS.ResourceAdequacy.merge!(
    a::SMCMyResultAccumulator, b::SMCMyResultAccumulator)
```

finalize!

Just as with the Convolution result spec, once all of the thread accumulators have been merged down to a single accumulator reflecting results from all of the threads, this final accumulator `acc` is mapped to the final result output through a `finalize` method.

```
function PRAS.ResourceAdequacy.finalize(
    acc::SMCMyResultAccumulator, sys::SystemModel)

    return MyResult(...)

end
```

Bibliography

Billinton, R. 1970. *Power System Reliability Evaluation*. Gordon / Breach.

Evans, M. P., S. H. Tindemans, and D. Angeli. 2019. “Minimizing Unserved Energy Using Heterogeneous Storage Units”. *IEEE Transactions on Power Systems* 34 (5): 3647–3656.

Haringa, G. E., G. A. Jordan, and L. L. Garver. 1991. “Application of Monte Carlo simulation to multi-area reliability evaluations”. *IEEE Computer Applications in Power* 4 (1): 21–25.

NERC Probabilistic Assessment Working Group. 2018. *Probabilistic Adequacy and Measures*. Tech. rep. North American Electric Reliability Corporation.

Zachary, S., and C. J. Dent. 2011. “Probability theory of capacity value of additional generation”. *Proc. IMechE Part O: J. Risk and Reliability* 226:33–43.