Getting the Most from Repairable Systems Simulation (RSS): Going Beyond the Documentation.

Repairable systems can be complex. The RSS platform offers several lesser known or undocumented features that can help users build simulations more closely aligned to how they are running a system. Did you know block replacement can be contingent on its age or the number of times it has failed? Or that maintenance schedules can be based on system state, making it possible to skip a maintenance if it would bring the system down? Using an example-based approach, this discussion will illustrate useful features and functionality not easily found or missing from the documentation and JMP User Community. Topics include getting, setting, and employing system information as well as applications for less commonly used RSS events and actions. A JMP Journal with all examples will be available.

There are six blocks available to represent system components: Basic, Series, Parallel, K out of N, Standby, and Stress Sharing. A seventh item, Knot, can be used to organize paths coming from or going to other blocks and to check the number of operational outgoing paths. It cannot take any of the other characteristics associated with blocks, such as events, actions, and distributions. For the sake of our discussion, we will not consider it a block. All blocks, except Basic, are composite, comprised of more than one subcomponent. If necessary, the word subcomponent or subunit will be used to differentiate what goes inside one of these block from the block itself. All composites except Series contain subcomponents in parallel with varying degrees of specialty. The table below shows the differences between composite blocks in terms of number of running components, when the block fails, and how the subcomponents age.

Block	# Components Running	Fails When	Component Aging
Series	All	One unit fails	Equal
Parallel	All	All units fail	Equal
K out of N	All	Fewer than K operational units remain	Equal
Standby	K	Fewer than K operational units remain Switch failure	Variable
Stress Sharing	All	All units Fail Switch failure	Variable

With Standby up to *k* subunits can be running leaving the remaining subunits as backups, ready to run when an operational subunit fails. The backups can be made to age with the block or only when in use. Additionally, each backup is associated with a switch with a given failure probability. If a switch fails, the next available subunit is accessed. Block failure occurs when either the last back-up or switch associated with it fails. Alternatively, a single switch can be used for all backups. Failure of the switch in this situation leads to block failure. Stress Sharing lets subcomponent reliability be a function of the number of operational subunits. By default (Basic Stress Sharing Type), subcomponents age at a rate proportional to the (inverse of the) number of operating subcomponents. This can be altered using a custom function. Stress Sharing employs a single switch mechanism similar to Standby.

Perhaps the most important characteristic about the composite blocks discussed above is that subcomponents cannot take events or actions and must be treated as identical entities. It is not

possible, for example, to perform maintenance on a failed backup while another backup is running. One must wait for the Standby block to fail or to reach a scheduled action before anything can be done. Even then, the entire composite must be operated on as a single unit not allowing for differential treatment of subcomponents. Additionally, information about subcomponents, such as how many have failed at a given time, is not available.

To circumvent these issues, Basic blocks can be arranged to duplicate the behavior of any of the composite blocks.

Example 1 – Basic Blocks in a K out of N Configuration

In the first example, a 3 out of 5 composite is built with Basic blocks plus a Knot and put in series with a 3 out of 5 K out of N block. When a block fails it is replaced with new taking 40 hours. The distribution associated with each block is Weibull(1,2000). It is important to point out that the distribution associated with any composite block applies to the block subcomponents and not to the entire block. For example, an *n* subcomponent Series block with an Exponential(λ) distribution will fail at a rate of λ/n relative to a single subcomponent block with the same distribution. The two k out of n configurations in the example are identical with one notable exception. In the custom composite, subcomponents are replaced on failure. In the case of a K out of N Block, replacement does not occur until the k+1 subcomponent has failed. Failed subcomponents remain so until the entire block fails.

Clicking on the green button below Start will run 100,000 year-long (8760 hour) simulations. The output table contains the simulation number, time of event/action, name of the block with which the event/action is associated (Subject), the name of the event/action (Predicate), the state of the block/system (State), and information as to whether system state was changed automatically, manually, unintentionally or the end of the simulation (Note). The values in the Subject, Predicate, and State columns are numeric with Value Labels. The Values Labels used for Subject and Predicate are taken from the labels given in the diagram. Manually labeling blocks, events, and actions allow these values to be used rather than the defaults. Additionally, there are 12 standard labels used for both Subject and Predicate:

Value	Label
-100	System
-99	Turn On System
-98	Turn Off System
-97	Turn Down System
-96	System Is Down
-95	System Is Up
-94	Error
-93	Next Event Proposal
-91	Turn On Block by System
-90	Turn Off Block by System
-89	Simulation
-88	Exception

There are 6 standard labels for State

Value	Label

0	Off
1	On
2	Removed
3	Down
4	Start
5	Finish

Running the simulation from Example 1 in the JMP Journal provided with this document will almost certainly show a marked difference between the two configurations. The script button below the example will summarize the data. In most cases the number of failures from the built-in block will outnumber the custom block by 100 to 1 or more.

There are 5 events and 13 actions. Basic is the only block that allows all events and actions. The table below shows which events/actions can be applied to which blocks.

		Basic	Series	Parallel	K out of N	Standby	Stress Sharing
Event	Block Failure	X	X	X	X	X	X
Event	Scheduled	X	X	X	X	X	X
Event	Inservice Based	X					
Event	System is Down	X	X	X	X	X	X
Event	Initialization	X	X	X	X	X	X
Action	Turn Off System	X	X	X	X	X	X
Action	Turn On System	X	X	X	X	X	X
Action	Replace with New	X	X	X	X	X	X
Action	Minimal Repair	X	X				
Action	Turn Off Block	X	X	X	X	X	X
Action	Turn On Block	X	X	X	X	X	X
Action	Remove Block	X	X	X	X	X	X
Action	Install New	X	X	X	X	X	X
Action	Install Used	X					
Action	Change Distribution	X					
Action	Inspect Failure	X	X	X	X	X	X
Action	If	X					
Action	Schedule	X	X	X	X	X	X

Items unique to Basic are in green. Minimal Repair is available to both Basic and Series. Initialization, Block Failure, System is Down can only appear once for a block. System is Down is triggered regardless if which block caused the system to go down. Turn On/Off System additionally turns on/off every block in the system. Install New, Install Used, and Change Distribution do not turn on the system automatically, unlike Replace with New or Minimal Repair. For these actions, the system must be restarted manually. The purpose of these actions is to allow other actions to occur prior to system restart.

Both events and actions must connect to actions. The number of connections and action can take is unlimited. Blocks must connect to other blocks (or a Knot).

Example 2 – Basic Blocks in a Standby Configuration

In this example, a two-subunit custom standby composite is built using Basic blocks and put in series with a built-in Standby block. Both contain two subunits with Exponential(500) distributions and a cold standby (back-ups do not age when not in use). All blocks are replaced with new on failure, taking eight hours. Two examples will be given, one with no switch (i.e., a switch with 0 probability of failure) and one with 90% switch reliability. Unlike the built-in Standby, we will replace the back-up with a new unit, taking eight hours, when the main unit is brought back online. This example will illustrate the use of the Initialization event, If action, and illustrate a way to manually position actions anywhere in the diagram.

Start with the diagram from Example 2 in the Journal. Each block has a Block Failure event tied to a Replace with New action. As mentioned above, the two subunits are in parallel. However, the back-up is to remain off while the main subunit is operational. Add an Initialization event followed by a Turn Block Off action to turn the back-up off at the start. Additionally, the back-up needs to be replaced and turned off when the main subunit comes back online. To do this, we must add Replace with New and Turn Off Block actions to the back-up when the main subunit comes back online after repair.

The primary way to add an action to a block is to select the block, then choose the action from the list appearing at the bottom of the RSS window. This will add an action square to the top right of the block. If one or more actions already exist, the square will be placed at the top of the stack of actions. Actions added in this fashion can't be moved. This placement is often not the most convenient location. Fortunately, an alternative method for adding actions exists that allows relocation anywhere in the diagram. To demonstrate, select either of the two existing actions from the backup. A plus will appear to the right of the action. Click on the plus and drag to a location away from a block or event. A pop-up menu will appear, select Replace with New. After a Replace with New action, the block is automatically started. Because we want to turn the back-up off after its replacement, we will add this action before turning off the block. Since we don't want the back-up replacement associated with either of the two current actions, select the connecting green arrow and delete it. Move the action square close to the Replace with New action for the main subunit and connect the two going from Replace with New for the main to Replace with New for the back-up. You can add the Turn Off Block action to the Replace with New action by dragging from it directly. Because it is associated with the back-up the added action will also be associated with the backup.

A drawback to moving actions in this fashion is that they can be easily confused as actions from the wrong block. Labeling blocks, events, and actions will help you to avoid this problem as the block an action is associated with appears in the Configuration information at the right.

To finish this example, we need to turn the back-up on when the main unit fails. The system will also need to be manually turned on. These actions can be added in a similar fashion to the Replace with New and Turn Off Block actions. Follow the Block Failure event from the main subunit to the Turn Block On, and from Turn Block On to Turn On System. In some, but not all situations, turning a block on will automatically turn the system on provided that an active path leading from

Start to End exists. When in doubt, turning on the system manually will not affect the results if the system is already on. Completed results can be found in the Journal (Example 2a).

Adding a switch with 90% reliability is relatively straightforward and makes use of the If action. Insert an If action between the Block Failure event for the main subunit and Turn On Block action for the back-up. In the If Condition script box put the code

```
Random Uniform() < 0.9
```

When this condition is not met, the actions placed after the If block are not executed. An If Condition can be any JSL expression. The last statement contained in it must, however, evaluate to true or false (any numeric value not 0 is considered true). If actions are used to incorporate conditional execution of actions. The completed Example 2 with the 90% reliable switch can be found in the Journal.

Example 3 – Two Pumps on a Staggered Maintenance Schedule

In this example, we will use Schedule and Initialization events and Schedule and If actions to set up maintenance for two pumps in parallel occurring every two weeks. The maintenance is staggered such that one pump receives it one week and the other pump the following week. This avoids having both pumps down for maintenance at the same time, causing the system to stop. Maintenance will take eight hours and bring the pumps to as new condition. If a pump fails, however, only a minimal repair is performed, taking four hours. Both pumps will have an Exponential(1000) distribution. In addition, since we don't want a planned maintenance taking the system down, we will have to check if the pump not undergoing maintenance is operational. To do this, we will make use of the Simulation Context variable where this information resides. The simulation will be run for one year.

Start with two pumps in parallel with a Block Failure event followed by a Minimal Repair action. This set-up can be found in the Journal, Example 3. To the pump getting the first maintenance, Pump A in the attached example, add the Scheduled event with Recurring Interval set to 336 (every two weeks) and follow it with an If action. Use the following code in the If Condition

```
Simulation Context["Status of Pump B"] == "Active"
```

`Simulation Context` is the variable, an associative array, where system information is stored. It can be read from, but not written to. To see what values are contained `Simulation Context` in you can use an If action with the condition:

```
Print(Simulation Context);
1;
```

Remember that the last statement in If Condition must be true or false. Removing the `1` would result in an error. Leave Completion Time for the If action to Immediate. Follow the If action with a Replace with New action (completion time 8 hours).

To set up the staggered schedule for the second pump, start with an Initialization event followed by a Schedule action. Set Max Occurrence to 1 and Completion Time to Constant/168. Follow this with another Schedule event with Max Occurrence set to missing (i.e., unlimited occurrences) and Completion Time Constant/336. These steps tell the system to wait one week from start, then schedule a regularly reoccurring event every two weeks. Follow the last Schedule event with an If, then Replace a with New event similar to those found for the first pump. In this case, the If Condition will be

```
Simulation Context["Status of Pump A"] == "Active"
```
since we are looking to see if the other pump is operational. This should complete the diagram. If either of the If Conditions are false, the maintenance is skipped. A completed version can be found in the Journal.

Example 4 – Custom Stress Sharing

The Stress Sharing block offers both basic and custom stress sharing. Aside from indicating that stress is shared across the subunits, there is little in the documentation explaining what this means. Additionally, there is no example demonstrating the use of the Custom option. This example will cover both of those cases in more detail.

The example in the Journal contains a system where three Stress Sharing blocks are put in parallel with no events. The topmost uses Basic for Stress Sharing Type, the middle uses Custom with the default Sharing Formula and the bottom uses a custom Sharing Formula. All blocks have five subunits with the Exponential(1) distribution. With this configuration, the system will run until all three blocks fail. Running this with a large number of simulations and looking at the distributions for the three blocks will show Basic Stress Sharing and Custom Stress Sharing 1 to be similar. Custom Stress Sharing 2 will have a short mean/median time, but a longer tail.

With Stress Sharing blocks, all components are active at the same time but, unlike Parallel, blocks ages differentially. Basic aging occurs at a rate of $1/k^{th}$ of that for a single component, where k is the number of active components. For example, if a block starts with five subunits, aging occurs at a rate of $1/5^{th}$ that of a single subunit. If the mean time to failure (MTTF) for a single subunit is *t*, then the MTTF for the first subunit in a 5-subunit Stress Sharing block will be $5\underline{t}$. When a component fails, the aging rate is updated to $1/4^{th}$ that of a single subunit. This continues until there is one subunit, which ages at the expected rate. The data table Show Stress Sharing illustrates this with a 5-subunit system.

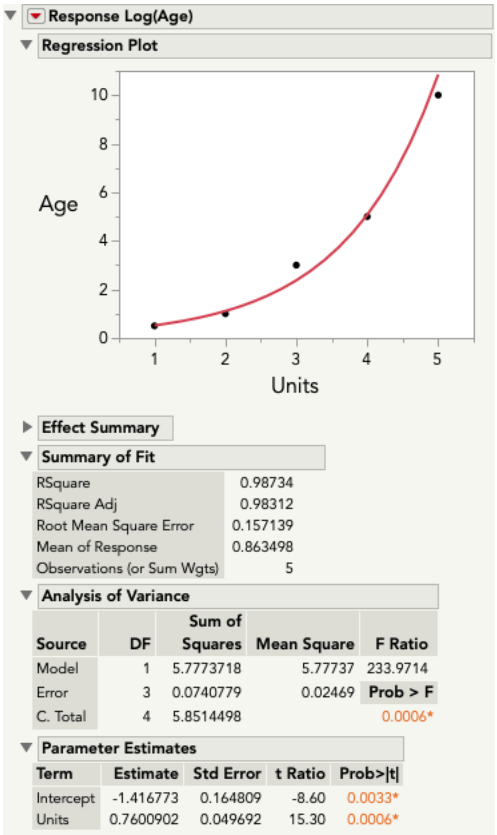
With this information, we can more accurately use the Custom Stress Sharing Type so that it reflects our intentions. Because of the mathematical functions used for the reliability distributions, the Log function is needed for the aging to occur in a linear fashion by default. One way to determine how to specify the custom function would be to first create a table of desired aging rates relative to the number of remaining components. Let's say, for a five-subunit block, we desired these aging rates:

Units	Age
5	10
4	5
3	3
2	1
1	0

We can use the `Choose` function:

```
Log(Choose(n,10,5,3,1,0.5))
```

Or model `Log(Age)` as a function of `Units`. Using Fit Model, this produces the results:



Saving the prediction formula to the data table and rounding, we get:

```
Exp(-1.42 + 0.76)
```

To finish, put this inside the Log function

```
Log(Exp(-1.42 + 0.76))
```

Example 5 – Nonparametric Distributions, Actions based on system age

In this example, we implement actions dependent on system age to simulate different failure distributions. If a failure occurs in the first two weeks (256 hours) the block will be repaired, taking 24 hours, and any early-stage failures removed, making the system more reliable. If the block fails between two weeks and 3000 hours, it will again be repaired (48 hours) removing any mid-stage failures and returning the system to an even more reliable state. Finally, if the block fails after 3000 hours, only minimal repair is performed with no change to the underlying reliability distribution. The changes in reliability will be modeling using the columns found in the data table Multi Failure Modes. The column All contains a mixture of all three failure types (early, mid, and late), Mid and Late contains mid and late failures, and Late Only, only the late failures. If actions will be used to check system age and Change Distribution to change the distribution. As mentioned above, Change Distribution requires a manual system restart. A completed diagram can be found in the Journal (Example 5).

To associate data from a table column to a nonparametric distribution, select the block and click the button with three dots to the right of Data in the configuration area. Click the Import button at the top left and select the table with the All column. If the table is not currently open, the Other… option will let you open a table. Select the column from the list box on the left and click Time to Event, then OK. When this is done, an Invisible copy of the table with all columns is saved to the

RSS script. All table properties, row states, and column properties are removed except Data Type, Modeling Type, and Format. Formulas are replaced by the actual values in the table. Depending on the number of columns in the data table, it may be best to create a subset table with only the column to be used. The censor value is saved in another part of the RSS script.

The diagram from the Journal contains only one event, Block Failure. From that, three If actions are used to determine the time period in which the failure occurs. `Simulation Context["Block Life of Processing Unit"]` is used in all three cases. Both the early and failure paths continue with Replace with New, Change Distribution, and Turn on System actions. The late failure path follows with a Minimal Repair action only.

Failure times for nonparametric distributions use random uniform deviates as quantile inputs for the nonparametric CDF. This is analogous to using the formula

`Col Quantile(`*nonParametricCol*`, Random Uniform())`

to generate the failure times, where *nonParametricCol* is the column containing the nonparametric data. Failure times are bounded by the maximum and minimum values in the data.

Example 6 – Convert a RBD to a RSS

Both Reliability Block Diagram (RBD) and RSS are saved as JSL scripts. Both begin with the line

```
//!
```

causing the script to execute rather than open. Because of this, they must be accessed by an alternative method, another application capable of reading text files (JSL files are ASCII) or by using one of the text reading JSL function such as `Load Text File`. Going from RBD to RSS is easier than going from RSS to RBD since all RBD blocks are also RSS blocks (RBD does not have Standby or Stress Sharing). Because a RBD can contain more than one diagram, its structure is somewhat different from that of RSS. Additionally, RBD can contain sub-diagrams, library elements, that are not found in RSS. To go from RBD to RSS (assuming there are no library elements):

1. Change `Reliability Block Diagram` to `Repairable Systems Simulation`.
2. Each `System Item` corresponds to a different diagram. Only one can appear for each `Repairable Systems Simulation`. Create one `Repairable Systems Simulation` for each `System Item`.
3. Delete all other arguments except `Show System Item`. The argument for `Show System Item` must match the first argument to `System Item`. A RBD will contain only one `Show System Item` argument. When working with more than one `System Item`, `Show System Item` will need to be added to those RSS diagrams that do not contain it.

While the results will not be identical to the JSL for a RSS, JMP will recognize and restructure it to its correct form.