RELIABILITY

TUTORIAL & REFERENCE

extendsim



ANDRITZ

© 2026 ANDRITZ Inc. This program is protected by US and international copyright laws.

You may not copy, transmit, or translate all or any part of this document in any form or by any means, electronic or mechanical, including photocopying, recording, or information storage and retrieval systems, for any purpose other than your personal use without the prior and express written permission of ANDRITZ Inc.

License, Software Copyright, Trademark, and Other Information

The software described in this manual is furnished under a separate license and warranty agreement. The software may be used or copied only in accordance with the terms of that agreement. Please note the following:

> ExtendSim blocks and components (including but not limited to icons, dialogs, and block code) are copyright © by ANDRITZ Inc. and/or its Licensors. ExtendSim blocks and components contain proprietary and/or trademark information. If you build blocks, and you use all or any portion of the blocks from the ExtendSim libraries in your blocks, or you include those ExtendSim blocks (or any of the code from those blocks) in your libraries, your right to sell, give away, or otherwise distribute your blocks and libraries is limited. In that case, you may only sell, give, or distribute such a block or library if the recipient has a valid license for the ExtendSim product from which you have derived your block(s) or block code. For more information, contact ANDRITZ at Info.ExtendSim@Andritz.com or Support.ExtendSim@Andritz.com.

© 2026 ANDRITZ Inc. This program is protected by US and international copyright laws. Microsoft is a registered trademark and Windows is a trademark of Microsoft Corporation. All other product names used in this manual are the trademarks of their respective owners. All other ExtendSim products and portions of products are copyright by ANDRITZ Inc. All right, title and interest, including, without limitation, all copyrights in the Software shall at all times remain the property of ANDRITZ Inc. or its Licensors.

Acknowledgments

Extend was created in 1987 by Bob Diamond; it was re-branded as ExtendSim in 2007.

The contents of this document are the result of years of work by software architects, simulation engineers, and technical writers and editors of ExtendSim products.

ANDRITZ Inc • 13560 Morris Road, Suite 1250 • Alpharetta, GA 30004 USA
408.365.0305 • Info.ExtendSim@Andritz.com
www.ExtendSim.com

Table of Contents

Introduction 1
Welcome! 1
About this document 1
Who should read this document 1
Chapters in this reference 2
Introduction to reliability block diagramming 2
Two types of discrete event tools 3
Advantages of integrating RBD with PSS 4
When to use the Reliability module 5
Framework of the Reliability module 5
Reliability module features 6
Where to get more information 7

Basics: Exploring an RBD 9
Overview 9
The bicycle 9
Exploring an RBD model of the bicycle 10
Structure of the model 13
Blocks used for the stand-alone RBD model 14
Descriptions of the blocks in the model 14
RBD databases 20
Results of running the Bicycle RBD 20
Next steps 21

Tutorial 1: Creating an RBD 23
Overview 23
Start a new model 23
Set the simulation parameters 24
Create an RBD 24
Distribution classes 26
Event cycle classes 30
Associate the event cycles with the nodes 34
Associate the interrupts 38
Availability 41
Conclusion 41
Run the model 41
Results 41
Next steps 43

Tutorial 2: Adding PSS to RBD 45
The tutorial models in this chapter 45
The 2A RBD Tutorial model 46
Change the event cycle progress type 47
Connect the process model to the RBD 48

Tutorial 3: Add Reliability to a Rate Model 51

Tutorial 4: Adding a Gantt Chart to an RBD Using Python 57

Reference 65

RBD terminology 65

Example models 66

Basics 67

..... 69

Reliability
Tutorial & Reference

Introduction

Welcome!

This guide introduces the tools and techniques used to model, analyze, and improve system reliability, maintainability, and availability. Whether you are new to reliability modeling or an experienced user, we hope this tutorial and reference guide helps you get the most from ExtendSim.

About this document

This document explores the ExtendSim Reliability module and shows how to use it as either:

- A standalone reliability block diagram (RBD) tool
- An RBD tool integrated with ExtendSim process simulation software (PSS) capabilities

Who should read this document

The availability of critical resources to perform work is often a key limiting factor in system performance. Yet identifying which resource availabilities are most important, and to what extent the timing and duration of their unavailability impacts the system, can be a complex problem to solve.

Since the ExtendSim Reliability module can be used as either a standalone RBD tool or in conjunction with ExtendSim simulation capabilities, this document will be helpful for:

- Anyone looking to explore the availability of an entire system and/or its individual resources.
- ExtendSim modelers using the Item and/or Rate libraries of ExtendSim Pro to simulate systems. Since the Reliability module seamlessly integrates with those libraries, you can explore the impact of resource availability on key process metrics such as throughput, production costs, repair costs, utilization, inventory, service levels, and so forth.

See also “Advantages of integrating RBD with PSS” on page 4 and “When to use the Reliability module” on page 5.



This document assumes you already know how to launch ExtendSim and build a discrete event model. If not, see either the Discrete Event or Discrete Rate Quick Start Guide (QSG). In addi-

tion, since building RBD's is so integrated with the ExtendSim database, we suggest you read the ExtendSim Database Tutorial and Reference.

Chapters in this reference

1) Introduction to reliability (this chapter of the document):
2) Basic information: exploring a reliability model
3) Tutorial 1: building a stand-alone RBD of a bicycle
4) Tutorial 2: adding a discrete event process to the RBD
5) Tutorial 3: adding an RBD to a discrete rate model
6) Reference: a comprehensive catalog of Reliability features and capabilities

Introduction to reliability block diagramming

✦ For an in-depth explanation of RAM (reliability, maintainability, and availability) and other concepts, see the last chapter of this document.

Reliability block diagramming (RBD) is:

- A methodology that graphically and statistically describes a system's resource availability over time as well as the impact it has on the system as a whole.
- A network of nodes that have been connected in series and/or in parallel, with one entry point into the network, a single direction through the network, and one exit point from the network.

Availability and down events

In the context of reliability, the term *availability* is defined as the percentage of time a resource is available to perform work.

Factors that would cause a resource to become unavailable are categorized as either scheduled or unscheduled down events:

- Scheduled down events represent planned downtimes for things like maintenance and off-shifting.
- Unscheduled down events represent downtimes due to unintended and unexpected interruptions (failures).

RBD

An RBD provides both a *graphical* and a *statistical* description of the availability of resources, capturing complex availability behavior for individual resources and for entire systems.

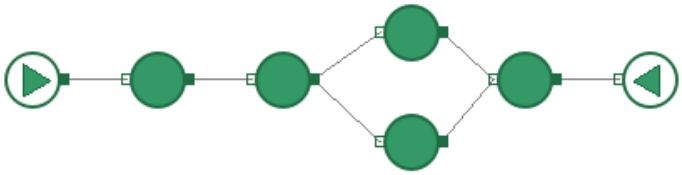
Graphical description

An RBD is a *directed acyclic graph* of nodes and edges where interior *nodes* (also known as components) represent resources that fluctuate between their up and down states while *edges* describe how the nodes are related to each other.

A system represented by an RBD can be as simple as a one-component network or as complex as a web of many nodes that have been placed both in series and in parallel.

- Components placed in *series* indicate that all the resources need to be in an up state in order for the system to be in an up state, i.e. available to perform work.

- Alternatively, components placed in *parallel* indicate redundancy. As shown in the RBD below, of the two parallel resources, only one needs to be up in order for the system to be available for work.



Statistical description

Since each node in the graph contains information about its probability of being in an available state, an RBD is also a *statistical representation* of the resources in the system. In other words, each node contains one or more statistically defined *event cycles* (also known as *fail modes*) that describe the resource's availability behavior over time.

Name[2]	Purpose[3]	Distribution Type[4]	Parameter 1 [5]	Parameter 2 [6]
Failure Brake	TTD	Weibull	80.00000	50.00000
Failure Drivetrain Chain	TTD	Weibull	90.00000	10.00000
Failure Drivetrain Crank	TTD	Weibull	365.00000	100.00000
Failure Drivetrain Derailleur	TTD	Weibull	90.00000	20.00000

Advantages of RBD

Using RBD to describe both the individual resource and the overall system availability has the following advantages:

1) **Visual Logic**. RBD's are really good at visually describing the relationships between resources. Instead of filling out tables by hand or writing code or logical statements, relationship logic can be captured with minimal effort in a visual graph.
2) **Powerful**. A high degree of complex reliability logic can be captured in an RBD without having to write code.
3) **Validation**. Complex reliability logic can be visually validated rather than having to sift through tables or decipher code.

Two types of discrete event tools

Discrete event tools model systems where the simulation clock moves forward in discrete chunks of time, when an event occurs. These tools can be divided into two groups:

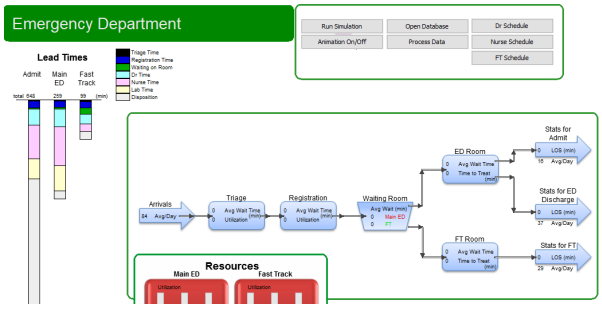
1) Process Simulation Software (PSS) tools, which model the dynamic behavior of the system, simulating the process steps by which systems transform inputs into outputs.
2) Dedicated RBD tools, which graphically and statistically describe when individual resources or entire systems of resources become available and unavailable over time.

Even though the RBD and PSS tools both employ discrete event modeling techniques to manage the simulation clock, the two technologies were historically developed to solve different kinds of problems and evolved independently of each other. Consequently, very little cross-

pollination of ideas and capabilities transpired between them. As a result PSS and RBD tools typically benefit from a different set of strengths and suffer from a different set of weaknesses.

Process simulation software

Discrete event process simulation tools (e.g. ExtendSim and its Item and Rate libraries) are ideally suited for capturing the detailed behavior of the complicated "process-based" types of systems that often occur in manufacturing, military operations, and logistics, such as the emergency department shown at right.



For example, assume you want to determine the ideal conveyor speed in a bottling plant that also uses manual labor. You would need to account for the impact that changing speed would have on other critical resources. One such resource might be the labor needed to restock consumables like stickers, caps, and bottles along the line. As conveyor speeds increase, how much more labor is needed to keep up? To answer that question, the PSS can literally map out the path walked by the workers and define the time needed to traverse that path using a statistical distribution that changes during the course of a shift based on when labor starts to get tired. Although you wouldn't want to "over-model" the system, these kinds of nitty-gritty details can be essential to understanding how the system currently works and to planning how it could or should work. On the other hand, process simulation tools struggle to capture the sometimes extremely complex nature of availability. They would typically be ill equipped to model the complexity of hundreds of dependent and independent fail modes for the bottling plant.

RBD tools

Dedicated RBD tools are very good at capturing availability over time, providing a sophisticated level of reliability analysis. As discussed on page 3, RBD tools can visually capture and validate complex reliability logic and the relationships between a system's resources.

Going back to the bottling conveyor example, each machine on a packing line could have more than 500 dependent and independent fail modes that are capable of bringing the packing line down. RBD tools can quickly and easily model those types of complex failure behaviors. However dedicated RBD tools lack the ability to simulate the details of essential system behaviors (such as conveyor speed or the contention for labor resources) and to model how those behaviors impact the system as a whole.

Advantages of integrating RBD with PSS

The Reliability module provides a Reliability Block Diagramming (RBD) tool that can also be integrated with the powerful process simulation capabilities of ExtendSim.

Integrating the ExtendSim RBD capability with its process modeling modules supports high fidelity modeling of processes characterized by random unscheduled downs and subsequent repairs. This integration provides a number of compelling advantages:

1) More accurate component wearing. The process model can be used to define when and what types of machine wearing is occurring on the resources in the RBD. This can lead to a more accurate assessment of when wear-based failures occur.

2) Detailed repair modeling. When a resource in the RBD fails, ExtendSim modeling capabilities can be used to break out the repair process in as much detail as needed:
   - Are the tools needed to make the repair currently available or are they allocated to other tasks?
   - What is the current level of spare parts inventory and supplier lead time?
   - Are the labor resources qualified to make the repair currently available or are they performing other work?
   - Should we preempt key resources and redirect them to this higher priority job?
   - If the resources required to make this repair are currently unavailable, should we outsource the job?
3) The RBD's ability to model when resources and/or entire systems are down can be used to impact the movement of material through the simulated process. This allows the modeler to explore the relationship between resource availability and system performance metrics like throughput, production costs, repair costs, utilization, inventory, service levels, etc.

When to use the Reliability module

The Reliability module can be used as either a stand-alone RBD tool or in conjunction with ExtendSim process simulation capabilities.

✦ In either instance, using reliability requires reliability-specific data to properly populate the RBD's. For example, you'll need to determine which distributions to use to categorize the resources' failures, repairs, and so forth.

As a stand-alone tool

Use the Reliability module on its own any time you want to explore the availability of individual resources and the overall availability of the system.

In conjunction with PSS

An RBD is primarily an *availability* tool. In those cases where resource availability critically impacts model results, the Reliability library will play an important role. Specifically, consider integrating Reliability into your PSS models when the system being modeled possesses the following characteristics:

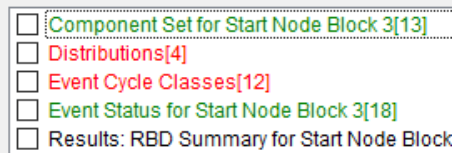
- Resource availability significantly impacts the model's key metrics.
- The availability status of a particular resource impacts the availability of other resources and/or the availability of the system as a whole.
- The system possesses key resources that go on/off shift, are prone to failure, and/or are taken off line for scheduled maintenance.

In systems such as these, the Reliability module will provide meaningful benefits to your PSS model building efforts and your post run analysis.

Framework of the Reliability module

The following components comprise the Reliability module.

1) ExtendSim **RBD databases**:



- Contain all the information needed to characterize the model's RBD's as you build them.
- Can be used to control RBD behaviors during the run.
- Store information that describes the current state of the RBD's.
- Document the results from model runs.
- Allow RBD's to easily scale up.

2) Nodes for creating the graphical structure of the RBD:



- **Start Node**. The first block in an RBD, it is in charge of scheduling down or up events for all components in its RBD. It is also responsible for determining the state of each individual node and the overall state of the RBD.



- **Component.** The interior of an RBD contains one or more Component blocks that represent components (in RBD talk) or resources (in PSS language), interchangeably. Failures, shift, maintenance, and other scheduled events are used to change component availability status over time.



- **End Node**. The terminating node in an RBD. During the run it reports the state of the RBD on its output connectors.

3) Other blocks:



- **Distribution Builder**. Creates classes of reliability-specific distributions and stores them in an RBD database. These distributions are used to specify the duration of time that an event cycle spends in its up or down state: time between downs (TBD), time to downs (TTD), and time to ups (TTU).



- **Event Builder**. Creates classes of event cycles from distribution instances and stores them in an RBD database. Event cycles control how components (resources) move between their up and down states over time.

✦ *Fail modes* is a common term used in RBD. To be more inclusive, the term *event cycles* will be used throughout this document to indicate shifts, maintenance, and other scheduled down events as well as failures.

Reliability module features

The ExtendSim Reliability module has many features that make it a powerful tool for modeling resource and system availability and determining how to better manage the resources in terms of redundancy and maintenance scheduling. In conjunction with a PSS tool, it is essential for analyzing the role resource availability plays in system performance.

This module includes:

- Graphical diagram/database builder. The Reliability module provides a graphical interface for building RBD's. As the user builds the diagram, the databases needed to support that diagram are automatically built.
- RBD/PSS interface. The Reliability module is fully integrated and supports communication between the RBD and PSS sections of the model.

- Distribution scaling. Since reliability distributions are stored in a generic format in the RBD databases, importing distributions from an external source (such as Excel) is supported. This means the number of distributions needed to support an RBD scales up easily.
- Multiple event cycles per node. To accurately model resource and system availability, any number of different event cycles such as failures, shifts, and maintenance events can be associated with a particular RBD node.
- Event cycle scaling. Since event cycles are stored in database tables, importing event cycles that have been defined in an external data source is supported. This means the number of event cycles needed to support an RBD scale up easily.
- Control logic. Optionally, the ExtendSim IDE can be used to write code to control all aspects of a node's behavior. For example, you can write code to control the speed at which a component progresses towards its next down based on any number of factors including the status of other related components, the state of the process model, seasonal policy changes, projected demand, and so forth.
- RBD databases. As mentioned earlier, these auto-built databases contain all the information needed to characterize the structure and current state of the RBD's in a model and document all the results from model runs. Additionally, these databases can be used to control RBD behavior during the run.

⚠ By default the ExtendSim Pro product allows a maximum of 100 event cycles/failure modes per model. As needed, additional event cycles can be purchased and added to ExtendSim Pro.

Where to get more information

The ExtendSim documentation, example models, and the video files and documents on the ExtendSim.com website provide comprehensive help.

Quick Start Guides

The purpose of a Quick Start Guide (QSG) is to get new users quickly familiar with a specific ExtendSim simulation methodology and aware of the ExtendSim features and capabilities. There are three Quick Start Guides—Continuous Process Modeling, Discrete Event Simulation, and Discrete Rate Modeling. Depending on the product purchased, one or more of these will be installed as eBooks in the Documents/ExtendSim/Documentation folder.

✦ It is recommended that you read either the Discrete Event or Discrete Rate Quick Start Guides before continuing with this document.

Tutorial & Reference documents

In addition to the Quick Start Guides, there are three Tutorial & Reference documents that are included as eBooks in the Documents/ExtendSim/Documentation folder:

- *ExtendSim Database*. This internal relational database provides model developers with a systematic way to manage information for the model and makes models scalable.
- *Advanced Resource Management*. Provides tutorials, examples, and reference information for using Advanced Resource Management capabilities in ExtendSim to model resource-constrained systems and their associated behaviors.
- *Reliability* (this document). Graphically capture and validate complex availability behavior. Determine when scheduled and unscheduled downs occur for individual resources and what impact that has on the availability of the system as a whole.

User Reference

The ExtendSim User Reference has a lot of information you will find helpful when building, using, and presenting models.

How To chapters cover general modeling and simulation topics

- Using libraries and blocks
- Performing analysis
- Enhancing presentations
- Creating a user interface
- Using equation-based blocks
- And much more

Appendices list menu commands and the ExtendSim libraries and blocks

Every menu command is explained; the main libraries are described block by block.

Technical Reference

You probably won't build your own ExtendSim blocks, but it's very common to use functions and logical statements in an Equation block (Value library) in a model. The Technical Reference lists over 1,000 functions and has information about using include files and other programming tools.

✦ The eBooks ship with the appropriate ExtendSim product. To access these documents, see the Documents/ExtendSim/Documentation/folder or launch the books from the Getting Started model that opens when ExtendSim launches. The User Reference and Technical Reference are also available if you select the Help menu when using ExtendSim.

Example models and videos show you how

ExtendSim includes numerous tutorial models as well as videos and example models that explain concepts discussed in the documentation.

Reliability
Tutorial & Reference

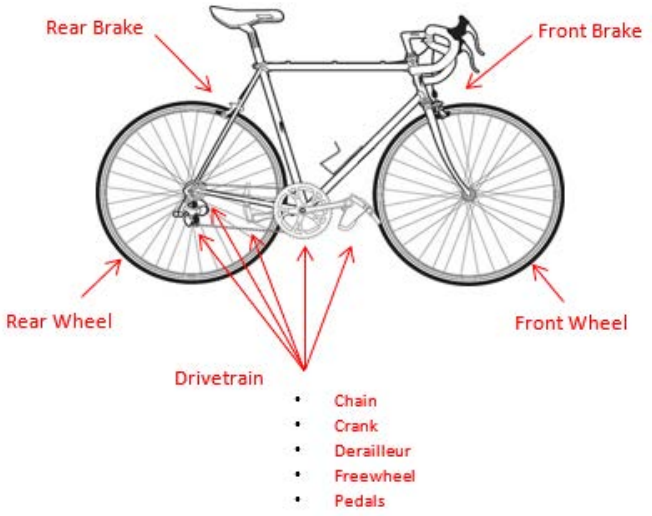
Basics: Exploring an RBD

Overview

This chapter discusses RBD terminology and describes RBD components while exploring an ExtendSim reliability model.

The bicycle

Assume a bike-messenger wants to model the reliability of her bicycle.



Bicycle components

As pictured above, the bicycle has the following components, each with their own failure rates:

1) Front wheel
2) Rear wheel
3) Front brake
4) Rear brake
5) A drivetrain composed of five subcomponents:
   - Chain

- Crank
- Dérailleur
- Freewheel
- Pedals

Assumptions

The bike-messenger:

- Has the tools and skills to make any repairs and has all the necessary spare parts on site, except for the Crank.
- Services the drivetrain annually. During this event, every subcomponent of the drivetrain, except the crank, is serviced.
- Services the front and rear wheels bi-annually.
- Can use the bicycle if either the front or rear brakes are working, but not if neither of them are working.

✦ For this model, the wearing of the bicycle components is not modeled explicitly. Instead, wearing is assumed to be occurring while the bicycle is in an up state as the simulation is running.

Why simulate the bicycle using RBD?

The purpose of running an RBD is to determine if the system (in this case the bicycle) is up and available for work or if and how often one or more downed components are conspiring to take the entire system down for some period of time. The bicycle is in an "up" state if one or more paths through the network are up; otherwise it is down.

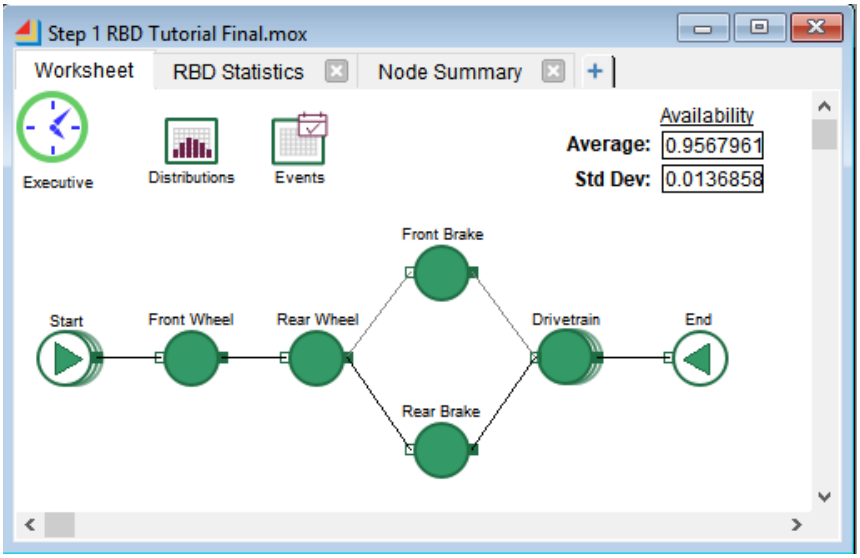
Exploring an RBD model of the bicycle

This section uses a model of the bicycle's reliability to describe an RBD and its components.

Open the example model

▸ Launch ExtendSim

▸ Open the model **Step 1 RBD Tutorial Final**; it is located at Documents/ExtendSim/Examples/Tutorials/Reliability.



The model worksheet

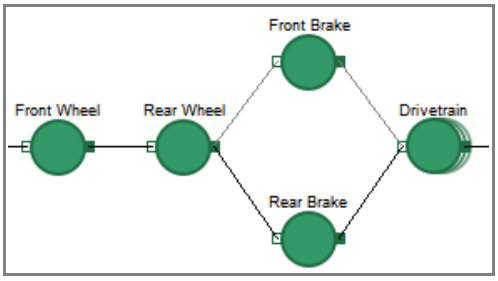
The model worksheet has ten icons, called blocks, seven of which are connected together.

- The set of seven connected blocks, starting with the *Start Node* and terminating at the *End Node*, is the RBD. The blocks between the Start and the End are *Components*.
- The other three blocks in the model (*Executive*, *Distribution Builder*, and *Event Builder*) perform specific modeling tasks in reliability models.
- The model also has cloned dialog items that report the average availability of the system as well as the standard deviation.

The blocks are discussed more starting on page 14.

Paths and redundancy

Notice that in this RBD the Front and Rear Wheels are in series but the Front and Rear Brakes are parallel to each other. The brakes provide two paths through the model—either the upper path that goes through the front brake or the lower path that goes through the rear brake.



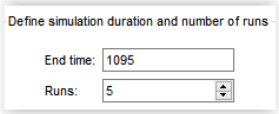
Placing the front and rear brakes in parallel to each other provides *redundancy*—if one of the brakes is down, there is still an alternate path through the RBD so the entire system isn't down.

Conversely, if even one of the other components (e.g. either the front or the rear wheel) is down, the system is down and the bicycle won't work.

Run the model

- Run the model by clicking the Run Simulation button or using the Run > Run Simulation command.

To explore variability, the model is set to run 5 times (run numbers 0-4) each time you give the Run command, and each run is set for 1,095 simulated days (3 years).

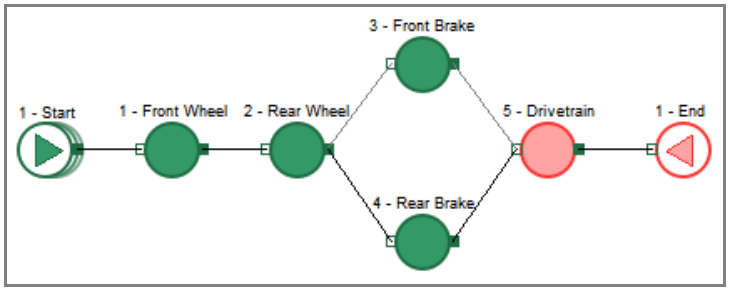


✦ To see status changes, run the simulation with animation on. If the animation runs too quickly for you to see what is happening, use the toolbar's Animation Slider to reduce the speed. Or click the Pause button (which replaces the Run Simulation button during the run), then click the Resume button to continue the run.

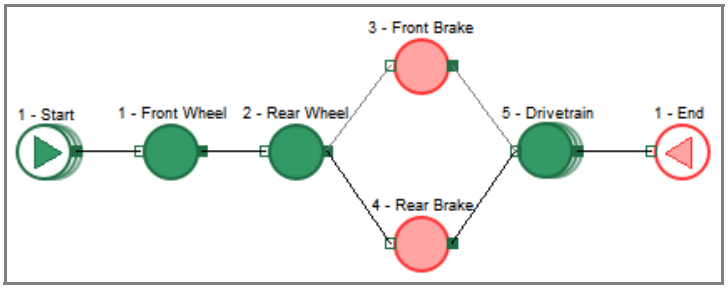
Watch the RBD system go up and down

If a Component is down for any period of time during the simulation, its icon will change from green to red. Whenever the End node is red, the entire system is down.

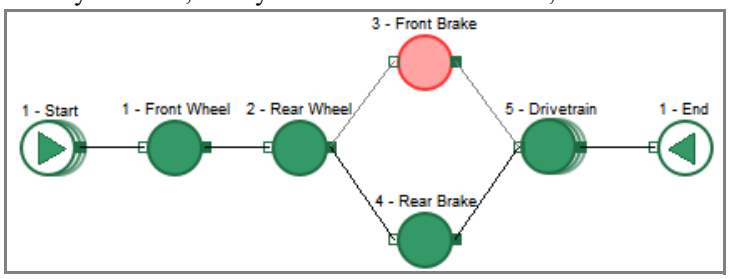
As shown below, the Drivetrain is down so the system is down, as reflected in the End node.



At the point in the simulation shown below, both brakes are down so the entire system is down.



Due to redundancy however, if only one of the brakes is down, the RBD is still up.



Structure of the model

Before exploring the results of running the RBD, it's a good idea to learn about model structure. A reliability model is composed of *blocks* on the model worksheet that provide the visible layout, or manage aspects of, the RBD, as well as a *database* that manages data "behind the scenes" and is saved with the model.

✦ This model only has one RBD but a reliability model can have multiple RBD's.

Blocks

Each of the icons on the model worksheet represents one *block*. A block is composed of an icon, a dialog that is accessed by double-clicking the icon, and code that determines how the block behaves over time as the model is run.

As seen in the screen capture on page 11, the model worksheet has 10 blocks:

- The seven *nodes* in this model (a Start Node, five Components, and an End Node) provide the actual layout of the Bicycle RBD. In this model, the Component blocks represent the mechanical parts of the bicycle described earlier.
- The Executive, Distribution Builder, and Event Builder blocks are used to manage aspects of the RBD, such as scheduling events. Every reliability model has one, and only one, of each of these blocks.

The model's blocks are discussed in more detail starting on page 14.

RBD Database

This model contains an ExtendSim database, named RBD, that can be accessed at the bottom of the Database menu. As RBD models are built, the database automatically stores and manages the reliability data for the model.

Blocks used for the stand-alone RBD model

There are 6 different blocks in the model of the bicycle RBD, described below.

Block	Library	Block Function	See Page
 Start Node	Reliability	Performs most of the RBD's critical tasks, manages the RBD's behavior over time, and can be thought of as the brain of the RBD. Can alternate between up and down states due to event cycles associated with class definitions in the Event Builder.	15
 Component	Reliability	Represents resources. Over time, Components exhibit life cycle behavior, alternating between up and down states due to event cycles associated with class definitions in the Event Builder.	16
 End Node	Reliability	Ends the RBD and reports its status.	16
 Executive	Item	Manages the events that have been scheduled by the Start Node. Required in every discrete event model.	17
 Event Builder	Reliability	Creates classes of event cycles for use by the Start Node and Component blocks to model an RBD's up and down states over time. The duration of these states is defined by instances of statistical distributions created in the Distribution Builder block.	17
 Distribution Builder	Reliability	Creates, stores, and provides classes of statistical distributions. These distributions define the duration of the up and down states defined in the Event Builder block: time-between-downs (TBD), time-to-down (TTD), and time-to-up (TTU).	19

Descriptions of the blocks in the model

✦ For a complete description of the blocks, see the last chapter of this document or the block's Help.

There is nothing fundamentally different about the structure of these different blocks. Any block may create, modify, or present information, and many blocks perform more than one of these functions.

Start Node

A model can have multiple RBD's; however, the Start Node is always the first node in each RBD. It performs most of the RBD's critical tasks, manages the RBD's behavior over time, and can be thought of as the brain of the RBD.

Icon

As with other blocks in the model, the Start Node has its ID and name above the icon. The ID is assigned automatically; the name can be changed in the block's dialog.

The icon of the Start Node has three modes. If the Start Node has:



1) None or 1 event cycles, the icon is a single circle as shown at the top of the screenshot at right
2) Two or more event cycles in *series*, the icon is stacked to the right as the middle icon shows
3) Two or more event cycles in *parallel*, the icon is stacked downward as seen in the icon at the bottom

✦ Any node with multiple event cycles will have its icon similarly stacked.

Functions

The Start Node is responsible for the following functions:

- Documenting input and output information in the RBD database
- Associating event cycle instances with nodes in the diagram
- Scheduling up and down events for all nodes in the diagram during the simulation run
- Calculating all paths through the diagram
- Collecting all run results for the entire diagram

Block dialog

▸ Double-click the icon of the Start Node to open its dialog.

✦ When its dialog is opened, a node's icon turns yellow; the icon stays yellow until the dialog is closed.

The Start Node has many tabs, which are described in detail later. For now:

▸ Go to the *Event Cycles* tab. Depending on the popup choice, this tab displays either all the event cycles that are associated with the RBD or just the event cycles for specific nodes.

▸ In the *Add/remove event cycle instances* frame, select **Show event cycles for: Front Brake**

Node	Event Name	Comp-DE Interrupts	RBD-DE Interrupts	Address	Edit Class
3-Front Brake	Brake Cycle	Preserve	Preserve	1:12:0:3	

As seen here, the Component representing the Front Brake has an ID number of 3 and it uses only one event cycle—the Brake Cycle (associated with the class definition in the Event Builder block). Information about the Brake Cycle is stored in the RBD database (number 1) at table 12, record 3.

Components

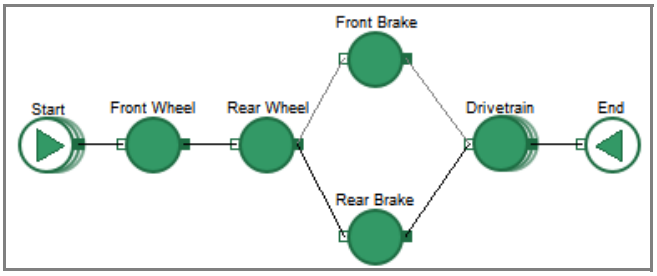
Every RBD must have at least one Component located between its Start Node and its End Node. Components represent resources and are placed in series and/or parallel to each other.



Over time, Components exhibit life cycle behavior, alternating between up and down states due to event cycles that are associated with event cycle classes defined in the Event Builder block.

The RBD's Components

As seen here, there are five Components in this RBD:



- Front Wheel
- Rear Wheel
- Front Brake
- Rear Brake
- Drivetrain

Each Component represents a mechanical part of the bicycle and each has its own event cycles to model its up/down behavior.

The Component for the Drivetrain represents the bicycle's five subcomponents—chain, crank, dérailleur, freewheel, and pedals—as discussed on page 9. Each of the subcomponents has its own event cycle to model its own up/down behavior. The Drivetrain icon is stacked to the right, indicating that the block has multiple event cycles in series.

✦ You can assign one or more event cycles directly to an individual Component in that Component's Event Cycle tab. However, if the event cycle would by definition cause the entire RBD to go down, it is best practice to assign that event cycle in the Start Node. For example, it is common to assign down events for maintenance in the Start node.

Block dialog

- Open the dialog of the Component labeled **Front Brake.**
- Go to the block's Event Cycles tab. Notice that the Brake Cycle event for the Front Brake, as shown in the Start Node block earlier, is also displayed in this dialog.

Node	Event Name	Comp-DE Interrupts	RBD-DE Interrupts	Address	Edit Class
3-Front Brake	Brake Cycle	Preserve	Preserve	1:12:0:3	

Dialog options

The tabs for a Component block (Around Me, Current State, etc.) are the same as for the Start Node block.

End Node

Each RBD must end at an End Node. The main purpose of the block is to indicate an end point for the RBD and to report RBD status on its output connectors. The General tab in its dialog displays the same information that is displayed in the General tab of the Start Node and Components.



Executive



The Executive block (Item library) does event scheduling and provides for simulation control, item allocation, attribute management, and more. Each reliability model must have one and only one Executive block, regardless of how many RBD's are in the model. The Executive must be present on the leftmost side of the model worksheet.

For a reliability model, the Executive manages the events that have been scheduled by the Start Node. Since the Executive handles this automatically, there are no settings to make in the dialog of the Executive block.

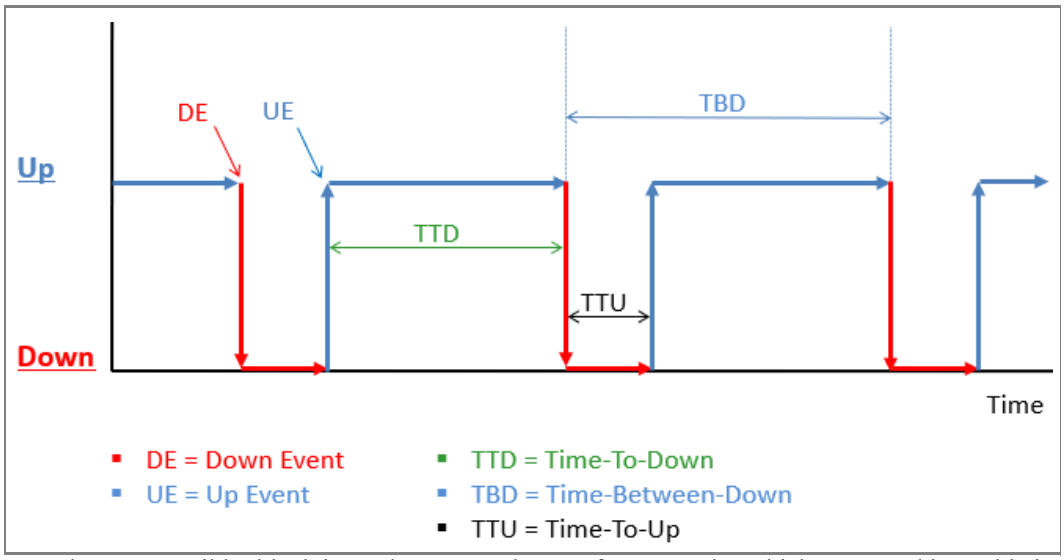
Event Builder



Each reliability model has one and only one Event Builder block (Reliability library), regardless of how many RBD's are in the model. It is used to define event cycle classes.

Event cycles

An *event cycle* is a never ending sequence of up/down state transitions (events). This cycling behavior is shown below.



The Event Builder block is used to create classes of event cycles which are stored in a table in the RBD database. The Start Node and Component blocks use instances of these event cycle classes to model their reliability life cycle behavior.

✦ The distributions that specify the duration of the up and down states are defined in the Distribution Builder block, discussed on page 19.

How the block works

Discrete events are the mechanism ExtendSim uses to cycle Start Node and Component blocks through their up and down states over time. No matter how many RBDs are in a model, the event cycle classes for an entire model are defined in the single Event Builder block.

When you add an Event Builder to the model, ExtendSim auto-creates an RBD database table named *Event Cycle Classes* that is maintained by the block.

As each event cycle class is created, it is stored in the Event Cycle Classes table as a class definition—one record for each class definition. Each class definition can have many instances in an RBD, where each instance is created when the class definition is associated with a particular event cycle. This allows for a class to be defined once but used in multiple event cycles.

✦ As will be seen in next chapter's tutorial, classes of event cycles can be defined in the Event Builder block or in an external application such as Excel, then imported into the model using the Event Cycle Classes database table.

The Event Builder block's dialog

▸ Double-click the Event Builder's icon to open its dialog.

The database

The top part of the Event Builder's dialog indicates that:

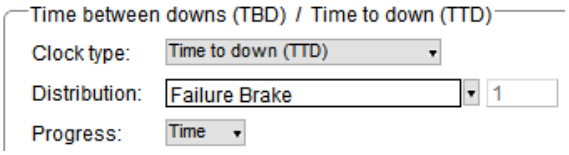
- The database for storing the Bicycle RBD's event cycle definitions is named *RBD*
- Definitions for the event cycles are stored in the database table named *Event Cycle Classes*
- The definitions of the distributions used to characterize the event cycles come from the *Distributions* table of the RBD database

The Brake Cycle event cycle

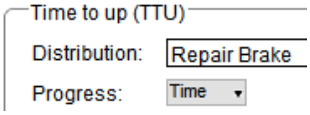
Each bicycle component has one or more corresponding event cycles that were created in the Event Builder block. To see the event cycle for the brakes:

▸ In the *Create/modify event cycle classes* dialog frame, choose **Event cycle name: Brake Cycle**. The Brake Cycle event is defined by two distributions:

1) A distribution that specifies how long the brakes will be up—the period of time from when the brakes recovered after the last failure until they will fail again (the *time to down* or TTD). For the brake this distribution is named *Failure Brake*. It is a Weibull distribution with Scale = 80, Shape = 50, and location = 0.



2) A second distribution that specifies how long it will take to repair the brake (the *time to up* or TTU). As shown here, the Brake Cycle uses a distribution named *Repair Brake*, a Normal distribution with a Mean of 7 and a standard deviation of 0.25.



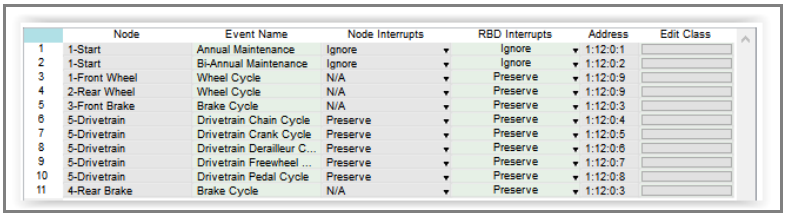
These distributions are instances of the distribution classes that have been defined in the Distribution Builder block, discussed below.

✦ There are three types of distribution classes that can be created in the Distribution Builder: time to down (TTD), time to up (TTU), and time between downs (TBD). A TBD-based event cycle schedules down events independent of repair and is typically used for calendar-based events such as Maintenance; it is described in detail later.

The bicycle RBD's event cycle classes

You can see the event cycle classes for all of the bicycle's components by accessing the Event Builder block's Event Cycle Classes database table or by looking in the dialog of the Start Node.

- Open the dialog of the Start Node and go to the Event Cycles tab
- In the *Add/remove event cycle instances* frame, select **Show event cycles for: All Nodes**.



	Node	Event Name	Node Interrupts	RBD Interrupts	Address	Edit Class
1	1-Start	Annual Maintenance	Ignore	Ignore	1:12:0:1	
2	1-Start	Bi-Annual Maintenance	Ignore	Ignore	1:12:0:2	
3	1-Front Wheel	Wheel Cycle	N/A	Preserve	1:12:0:9	
4	2-Rear Wheel	Wheel Cycle	N/A	Preserve	1:12:0:9	
5	3-Front Brake	Brake Cycle	N/A	Preserve	1:12:0:3	
6	5-Drivetrain	Drivetrain Chain Cycle	Preserve	Preserve	1:12:0:4	
7	5-Drivetrain	Drivetrain Crank Cycle	Preserve	Preserve	1:12:0:5	
8	5-Drivetrain	Drivetrain Derailleur C...	Preserve	Preserve	1:12:0:6	
9	5-Drivetrain	Drivetrain Freewheel ...	Preserve	Preserve	1:12:0:7	
10	5-Drivetrain	Drivetrain Pedal Cycle	Preserve	Preserve	1:12:0:8	
11	4-Rear Brake	Brake Cycle	N/A	Preserve	1:12:0:3	

This table lists all the event cycle instances for the nodes in the RBD.

Distribution Builder



Each reliability model has one Distribution Builder block (Reliability library) which is used to create, store, and provide statistical distribution definitions for time-between-downs (TBD), time-to-down (TTD), and time-to-up (TTU). The Event Builder block uses those definitions when creating event cycles.

How it works

No matter how many RBD's are in a model, all the distributions for the model are defined in a single Distribution Builder block. When you add a Distribution Builder to the model, Extend-Sim auto-creates an RBD database table named *Distributions* that is maintained by the block.

As distributions are created, they are stored in the database table as class definitions—one record for each class definition. Each class definition can have many instances, where each instance is created when the class definition is associated with a particular event cycle. This allows for a distribution to be defined once but used by the Event Builder in multiple event cycles.

The block's dialog

- Double-click the Distribution Builder block's icon to open its dialog.

The Distribution Builder block is for creating and modifying classes of distributions for use when creating event cycles in the Event Builder block.

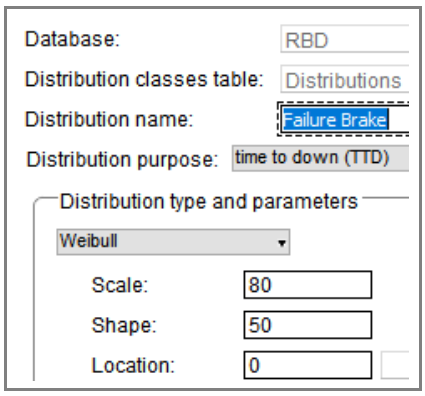
The database

- The top part of the block's dialog indicates that:
  - The database for storing distribution definitions is named *RBD*
  - The definitions of the distributions are stored the *Distributions* table of the RBD database

A distribution

This is where the distribution classes for the Brake Cycle were created. To view an existing distribution:

- In the *Create/modify distribution classes* frame, select **Distribution name: Failure Brake**. Its parameters are shown here.



✦ As will be seen in the tutorial, classes of distributions can also be defined in an external application, such as Excel, and imported into the model using the Distributions database table.

The Bicycle RBD's distribution definitions

To see all the distribution classes for the Bicycle RBD:

- In the Distribution Builder's dialog, click the **Open** button at the right of the Distributions field.



A portion of the database table that stores the distribution definitions for the Step 1 model is shown below.

	ID[1]	Name[2]	Purpose[3]	Distribution Type[4]	Parameter 1[5]	Parameter 2[6]
1	*TTD - Failure Brake*	Failure Brake	TTD	Weibull	80.00000	50.00000
2	*TTD - Failure Drivetrain Chain*	Failure Drivetrain Chain	TTD	Weibull	90.00000	10.00000
3	*TTD - Failure Drivetrain Crank*	Failure Drivetrain Crank	TTD	Weibull	365.00000	100.00000
4	*TTD - Failure Drivetrain Derailleur*	Failure Drivetrain Derailleur	TTD	Weibull	90.00000	20.00000
5	*TTD - Failure Drivetrain Pedal*	Failure Drivetrain Pedal	TTD	Weibull	365.00000	100.00000
6	*TTD - Failure Drivetrain Freewheel*	Failure Drivetrain Freewheel	TTD	Weibull	365.00000	100.00000
7	*TTD - Failure Wheel*	Failure Wheel	TTD	Weibull	180.00000	100.00000
8	*TTU - Repair Brake*	Repair Brake	TTU	Normal	7.00000	0.25000

RBD databases

While this model was being created, an RBD database was automatically created and populated with data. The structure, current state, and results from running the model get stored in tables in this database.

For example, the Distribution Builder was used to specify random distributions for the TTD's, TBD's, and TTU's for each component. They are stored in the database's Distributions table as shown above.

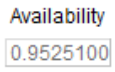
Results of running the Bicycle RBD

- Run the simulation. As noted earlier, the model is set to run five times, for a simulated duration of 3 years (1095 days) each run.

When it is run, a reliability model can generate a lot of information as you will see in the next chapter. For now, just look at availability.

Availability

At the end of the simulation run, notice that the cloned dialog item (top right side of the model) indicates that the average availability of the entire RBD was just short of 100%. (The distributions are random so your numbers will be slightly different than shown here.)



Note however that the availability number is essentially meaningless because the bicycle messenger doesn't know what the impact on her business will be. For example, the stand-alone RBD doesn't report how long it takes to deliver messages, how many messages were undelivered, what the revenues and costs are, and so forth. However, by integrating this stand-alone RBD with a process model, she can start to understand what the availability really means. You'll see that in the Tutorial 2.

Next steps

The next chapter, Tutorial 1, shows how to build the RBD you've just explored. That's lot more fun than reading about it!

Reliability
Tutorial & Reference

Tutorial 1: Creating an RBD

Overview

This chapter shows how to build the stand-alone bicycle RBD shown in the previous chapter.

⚠ The tutorials assume you know how to launch ExtendSim, open a library, place a library block on the model worksheet, and connect blocks. If you don't already know how, see either the Discrete Event or Discrete Rate Quick Start Guide.

Steps

1) Open a new model worksheet, then add and connect blocks to create an RBD
2) Create statistical distribution classes to define the duration for TBD's, TTU's, and TTD's
3) Create event cycle classes based on the distribution classes
4) Associate the event cycles with the nodes
5) Run the RBD

Start a new model

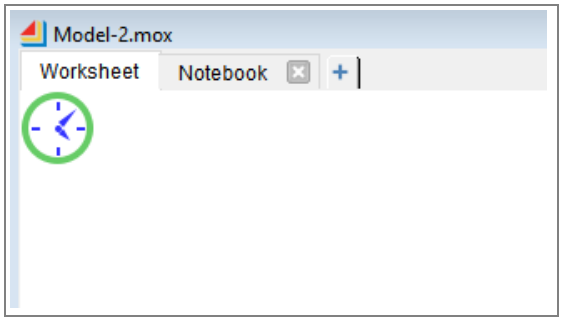
- Launch ExtendSim

By default, when you launch ExtendSim the Getting Started model opens as well as the major ExtendSim libraries and their library windows. You can close the Getting Started model.

Open a new model worksheet

- Use the toolbar button or the File menu to open a new model.

By default the model opens with an Executive block on the left side, as shown here.

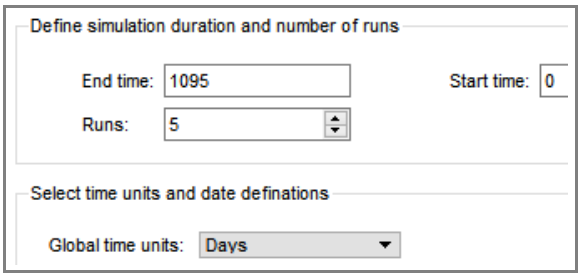


✦ Opening the major libraries on launch and inserting an Executive on the worksheet are default options in the Edit > Options menu.

Set the simulation parameters

The Simulation Setup command opens a window for entering global settings for the model, such as how long and how many times the simulation will run.

- Select the command Run > Simulation Setup
- In the dialog's Setup tab, enter the simulation parameters:
  - **End time: 1095**
  - **Start time: 0** (default)
  - **Runs: 5**
  - **Global time units: Days**



- Leave the other Simulation Setup settings at their defaults
- Click **OK** to close the window

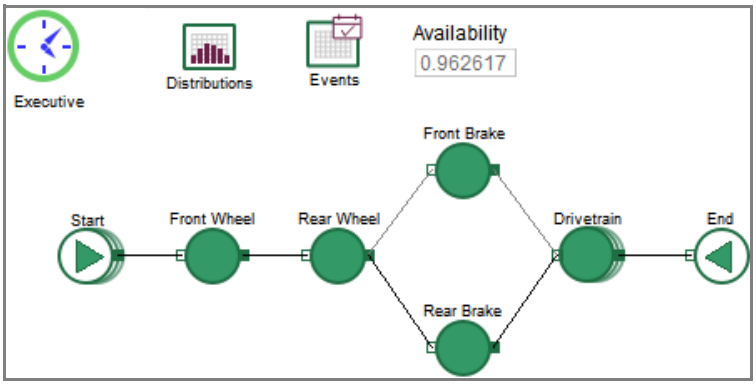
With these settings, the model will run five times, each for a simulated time of 1,095 days.

Save the model

- Choose File > Save Model As and name the file *My RBD*.

Create an RBD

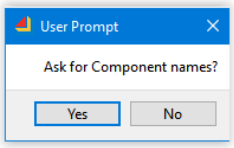
The goal is to create the model shown below.



Add the first three blocks

- Go to the **Reliability** library
- Place the following block on the worksheet:

1) **Start Node**. When the Start Node asks if it should ask for Component names, say **Yes**.



✦ After asking about the Component names, the Start Node should automatically place the Event Builder and Distribution Builder on the worksheet. If so, skip the next two steps.

2)**Event Builder**. If it isn't already on the worksheet, add it. Note that if you place this block on the worksheet before placing the Start Node and Distribution Builder block, ExtendSim will automatically put the Distribution Builder on the worksheet as well.

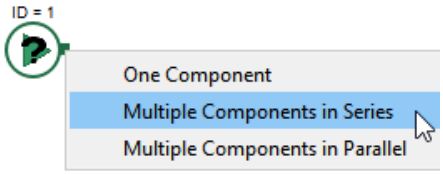
3)**Distribution Builder**. If it isn't already there, place this block on the worksheet.

No matter how many RBDs a model has, there must only one Event Builder and only one Distribution Builder.

✦ **Don't add any of the other nodes yet!** You can always manually add, connect, and name the nodes in your RBD. However, using the right-click connect procedure shown below is easier and more fun.

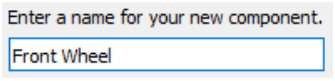
Add the first two Components

As shown in the model worksheet on page 24, the first two Components in your model should be in series.



- To add the first two Components, **right-click on the Start Node's output connector**
- In the dialog that appears, select **Multiple Components in Series**
- In the next dialog, choose to add **2** Components (the default option)

As ExtendSim adds each Component to the model worksheet, it asks you to enter a name for the node:

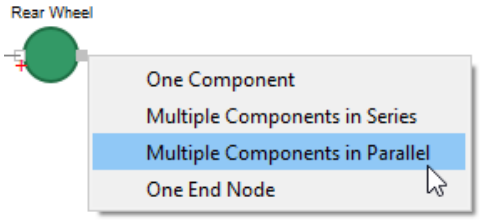


- Enter **Front Wheel** for the first Component
- Enter **Rear Wheel** for the second Component

✦ Notice that there is now a question mark on the Start Node's icon. It will stay that way until the layout of the RBD has been completed.

Add the second set of components

According to the model assumptions, the next two Components (the Front and Rear Brakes) are in parallel with each other:

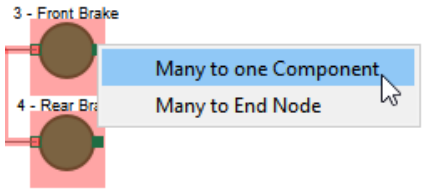


- To add the two Components in parallel, **right-click on the Rear Wheel's output connector**
- In the dialog that appears, select **Multiple Components in Parallel**
- In the next dialog, choose to add **2** Components
- When asked for the names of these new Components:
  - Enter **Front Brake** for the first Component
  - Enter **Rear Brake** for the second Component

Add the Drivetrain

The final Component is the Drivetrain.

- Select the icons for **both the Front Brake and Rear Brake**, as shown below
- *With both icons selected*, right-click on the output connector of **either** the Front Brake or Rear Brake
- From the dialog, select **Many to one Component**
- Name the new Component **Drivetrain**



Add the End Node

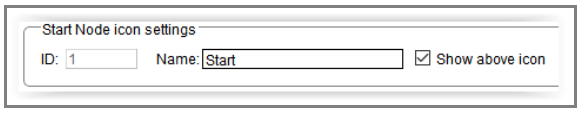
The last step in building an RBD is to add an End Node:

- Right-click on the Drivetrain's output connector and select **End Node**
- Save your model

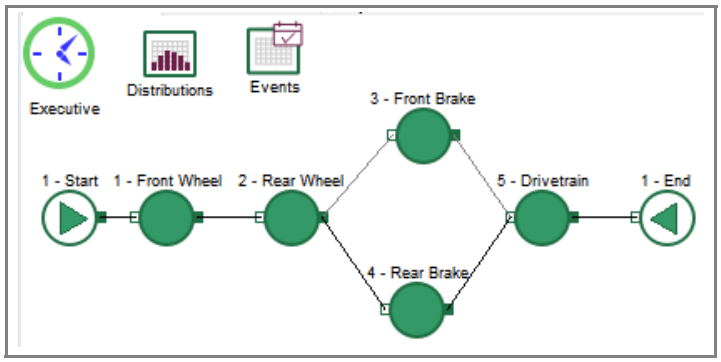
At this point the RBD layout is complete and the question mark on the Start Node is gone.

Name the blocks

- Use the label field next to each block's Help button to label these 3 blocks:
  - Executive
  - Distributions
  - Events
- The names for the **Start Node** and **End Node** must be usable by the database. To enable this, enter their names in the top frame of each dialog's General tab, as shown here.



When finished labeling, your model should look similar to this.



- Save your model

Distribution classes

The Distribution Builder is used to define different classes of statistical distributions. The definitions are stored in a Distributions database table that is automatically created and maintained by the block. Each record in the database table is a class definition for a particular distribution and each definition can have multiple instances when the class definition is associated with event cycles.

In this model, the event cycles primarily use random distributions to define the duration for the event cycles: TBD's (time between downs), TTU's (time to up), and TTD's (time to down). This section uses distribution classes than have been created in Excel and discusses how to make those distributions available to the Event Builder for use in creating event cycles.

Creating a distribution class using the Distributor Builder

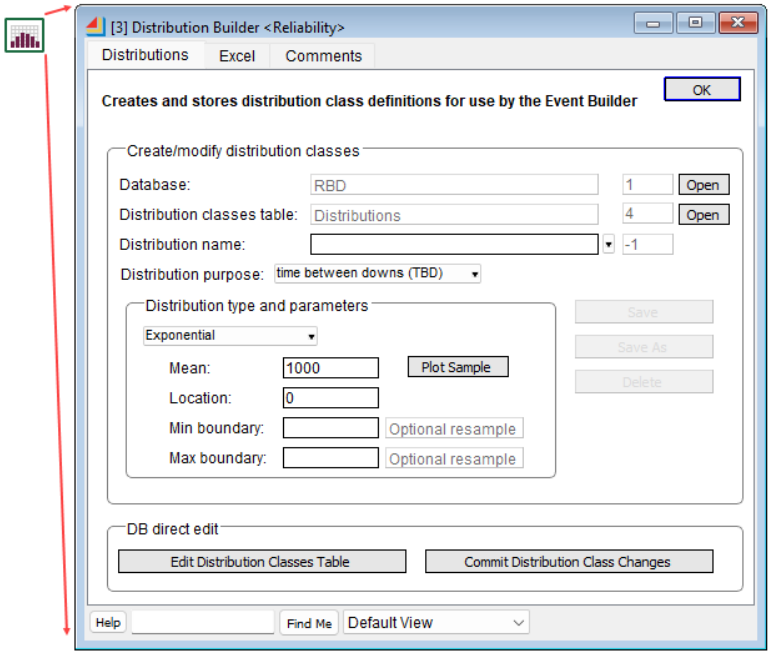
If you only need one or two distributions, you can define them directly in the Distribution Builder dialog. To do this:

- Open the **Distribution Builder** dialog.
- Enter a **Distribution Name**.
- Select the **Distribution Purpose** (TBD, TTD, or TTU).
  - **TBD** (**Time Between Downs**) for the operating time between failures.
  - **TTD** (**Time To Down**) for the time until a failure occurs.
  - **TTU** (**Time To Up**) for the repair or recovery time.
- Choose a statistical distribution type.
- Enter the distribution parameters.
- Save the definition.
  - The definition is stored in the **Distributions** database table maintained by the Distribution Builder.
  - Each record represents one distribution class definition.

The new distribution class will then be available for use by the Event Builder when creating event cycle classes.

Add a Distributor Builder block

- Place a **Distribution Builder** block (Reliability library) on the worksheet.
- Each Reliability model has a single Distribution Builder that stores all distribution class definitions used by the model.



Open the Distribution Builder dialog

- Double-click the Distribution Builder block to open its dialog.
- Distribution classes are defined in the dialog's Create/modify distributions section.

Create a new distribution class

- Enter a unique **Distribution Name** by clicking the down arrow to the right of the field and selecting **New Distribution**.
- Select the appropriate **Distribution Purpose**:
- Select the desired statistical distribution (such as **Weibull**, **Normal**, **Exponential**, **Lognormal**, or **Constant**).

Define the distribution parameters

- Enter the parameters required by the selected distribution type.
- The parameters displayed will vary depending on the distribution selected.
- Specify the appropriate time units for the distribution.
- Review the distribution graph and statistics, if desired, to verify that the distribution behaves as expected.

Save the distribution class

- Click the appropriate save/update control in the Distribution Builder dialog.
- The definition is stored in the **Distributions** database table maintained by the Distribution Builder.
- Each record represents one distribution class definition.

Use the distribution you just created

- Open the **Event Builder** dialog.
- Create a new Event Cycle Class as described in Event cycle classes on page 28 or edit an existing one.
- Select the new distribution from the TBD, TTD, or TTU distribution lists as appropriate.
- Save the event cycle definition.

Example

To create a distribution for annual maintenance:

- Create a new distribution.
- Distribution Name = **Annual Maintenance Time**
- Distribution Purpose = **TBD**
- Distribution Type = **Constant**
- Value = **365 days**
- Click **Save**

The distribution can now be selected in the Event Builder when defining an **Annual Maintenance** event cycle.

Copying the definitions into the model

If you were only going to use one or two distributions, it would be easiest to just define them in the dialog of the Distribution Builder. However, it is more likely that you will want many different distributions. The best way to do that is to define the distributions in an external application, such as done here in Excel, then copy them into a database table in ExtendSim.

Copy the definitions from Excel

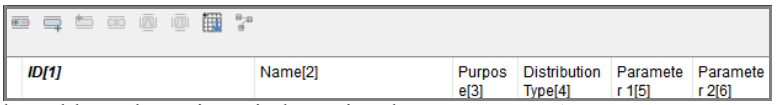
- Launch Excel and locate and open the Excel file named **Data for RBD Tutorial.xlsx**. The file is located at Documents/ExtendSim/Examples/Tutorials/Reliability.
- Go to the workbook's **Distributions** worksheet, a portion of which is shown below.

NAME	PURPOSE	DISTRIBUTION	PARAM 1	PARAM 2	PARAM 3	PARAM 4
Failure Brake	TTD	Weibull	80	50	0	0
Failure Drivetrain Chain	TTD	Weibull	90	10	0	0
Failure Drivetrain Crank	TTD	Weibull	365	100	0	0
Failure Drivetrain Derailleur	TTD	Weibull	90	20	0	0
Failure Drivetrain Pedal	TTD	Weibull	365	100	0	0
Failure Drivetrain Freewheel	TTD	Weibull	365	100	0	0
Failure Wheel	TTD	Weibull	180	100	0	0
Repair Brake	TTU	Normal	7	0.25	0	0
Repair Drivetrain Chain	TTU	Normal	0.25	0.1	0	0

- Copy all the cells from **rows 2 - 19** and **columns A - J only**. DO NOT copy the row numbers or the headers.

Open the Distributions database table

- In ExtendSim, open the dialog of the *Distribution Builder* block
- At the bottom of the dialog, click the **Edit Distribution Classes Table** button; this opens the Distributions table of the RBD database as shown below.



ID[1]	Name[2]	Purpose[3]	Distribution Type[4]	Parameter 1[5]	Parameter 2[6]

- With the Table as the active window, give the **Database > Append New Records** command or click the **Append New Records** button in the table's toolbar
- In the Append Records dialog, enter the number **18** and click **OK**; this results in a database table that can hold 18 records

Paste the definitions into the Distribution Builder

Notice that the Distribution table's first column (ID) is pink. Do not paste any data into that first column as it is reserved for internal use.

- In the database table, right-click in the cell of the **second column (Name)**, at **row 1**
- Give the **Paste/Paste Cells** command.
  - You should get a message reminding you to click the Commit Distribution Class Changes button after the definitions have been copied into the Distribution Builder.
  - Pasting causes all the definitions to be copied into that database table, as shown here.

ID[1]	Name[2]	Purpose[3]	Distribution Type[4]	**Parameter 1[5]**	Parameter 2[6]	Parameter 3[7]	Parameter 4[8]
	Failure Brake	TTD	Weibull	80.00000	50.00000	0.00000	0.00000
	Failure Drivetrain Chain	TTD	Weibull	90.00000	10.00000	0.00000	0.00000
	Failure Drivetrain Crank	TTD	Weibull	365.00000	100.00000	0.00000	0.00000
	Failure Drivetrain Derailleur	TTD	Weibull	90.00000	20.00000	0.00000	0.00000
	Failure Drivetrain Pedal	TTD	Weibull	365.00000	100.00000	0.00000	0.00000
	Failure Drivetrain Freewheel	TTD	Weibull	365.00000	100.00000	0.00000	0.00000

- If you have enough room on the monitor, leave the database table open to see the changes as you go through the next steps. Otherwise, you can close the table.
- At the bottom of the Distribution Builder's dialog, click the **Commit Distribution Class Changes** button. This saves the data to the RBD's other data structures.



- At the right of the *Distribution classes table* field, click the **Open** button to verify that the ID field (the pink column on the left) has been written to as shown below, then close the Distributions database table.

	ID[1]	Name[2]	Purpose[3]	Distribution Type[4]	Parameter 1[5]
1	*TTD - Failure Brake*	Failure Brake	TTD	Weibull	80.00000
2	*TTD - Failure Drivetrain Chain*	Failure Drivetrain Chain	TTD	Weibull	90.00000
3	*TTD - Failure Drivetrain Crank*	Failure Drivetrain Crank	TTD	Weibull	365.00000



Use the Commit button anytime you add records to the database. If you don't click the Commit button, the ID field (the pink column) won't be updated, causing database errors.

- Close the Distribution Builder's dialog
- Save your model; this saves the database changes with your model

The distribution classes have now been entered and the distributions will be available for use by the Event Builder.

Event cycle classes

While a Distribution Class defines statistical behavior, an Event Cycle Class uses those Distribution Classes to define how a component transitions between its up and down states. The Event Builder is used to define different classes of event cycles, which are discussed more fully on page 68. The definitions are stored in an Event Cycles database table that is automatically created and maintained by the block. Each record in the database table is a class definition for a particular event cycle and each definition can have multiple instances when it is associated with a particular Start Node or Component block. This allows for an event cycle to be defined once but used multiple times.

Event cycle table

As was true for the Distribution Builder, if you only need a few event cycles, creating them directly in the Event Builder is often the fastest approach. However, in most cases you will

have multiple event cycles, and it would be more efficient to define them in an external application such as Excel and then imported into ExtendSim.

Creating an event cycle class using the Event Builder UI

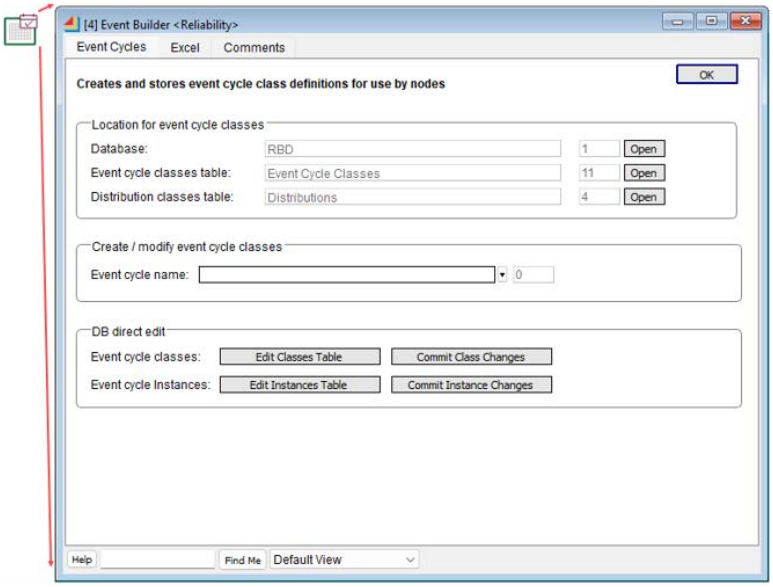
If you only need one or two event cycles you could just define them manually in the dialog of the Event Builder. To do this:

- Open the **Event Builder** dialog
- Enter an **Event Cycle Name**
- Select the **TTD/TBD** and **TTU** distributions from the Distribution Builder
- Save the definition
- Click Commit Class Changes

The new class will then be available for use by Start Nodes and Components.

Add an Event Builder block

- Place an **Event Builder** block (Reliability library) on the worksheet. Note, placing an Event Builder block to a worksheet will also automatically place a Distribution Builder block to the worksheet if there's not already one there.
- Each Reliability model has a single Event Builder that stores all event cycle class definitions.



Open the Event Builder dialog

- Double-click the Event Builder block to open its dialog.
- Event cycle classes are defined in the dialog's Event Cycles tab in the **Create/modify event cycle classes** section.

Create a new event cycle class

- Enter a unique **Event Cycle Name** (for example, Pump Failure, Annual Maintenance, or Brake Cycle) by clicking the down arrow to the right of the field and selecting New Event Cycle Class.
- Select the appropriate **Event cycle type** (Distribution, Shift, or 'Sched' connector).

- To define up/down behavior, choose **Distribution**, then click **Save**.

Defining the up/down behavior using a Distribution

- Choose the distribution type that controls the cycle:
  - **TTD** (**Time To Down**) or **TBD** (**Time Between Downs**) for the up period. For failure-type cycles, use the TBD/TTD setting (for example, Time or Wear/Usage, depending on the application).
  - **TTU** (**Time-to-Up**) for the down period for Time or Item progression.
- These distributions must already exist in the **Distribution Builder** and are selected by name within the Event Builder. Event cycle classes are built from distribution class instances.

Save the event cycle class

- Click the appropriate save/update control in the Event Builder dialog.
- The definition is stored in the **Event Cycle Classes** database table maintained by the Event Builder. Each record represents one event cycle class definition.

Commit the changes

- Click Commit Class Changes in the DB direct edit section.
- This updates the Reliability data structures and writes the ECCP Record Index values. If records are added but not committed, database errors can occur.

Use the class you just created

- Open a **Start Node** or **Component**.
- Go to the **Event Cycles** tab.
- Click **Add** in the **Add/remove event cycles instances** in the table and choose the new event cycle class.
- Set the appropriate interrupt behavior (Preserve, Reset, or Ignore) as needed.

Example

To create an **Annual Maintenance** event cycle:

- Create a TBD distribution called Annual Maintenance Time in Distribution Builder.
- Create a TTU distribution representing maintenance duration.
- In Event Builder:
  - Event Cycle Name = **Annual Maintenance**
  - Event Type = Distribution
  - TBD Distribution = Annual Maintenance Time
  - TTU Distribution = your maintenance-duration distribution
- Commit the class changes.
- Add **Annual Maintenance** to the Start Node's Event Cycles tab

Copy the definitions from Excel

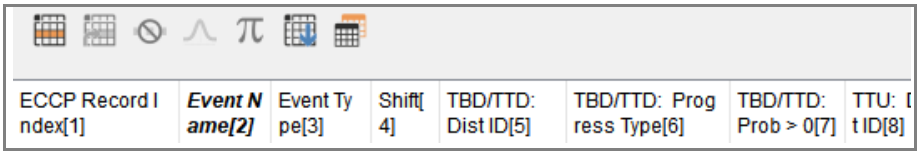
- If it isn't already open, launch Excel then locate and open the Excel file named **Data for RBD Tutorials.xlsx**. The file is located at Documents/ExtendSim/Examples/Tutorials/Reliability.
- Go to the Excel workbook's **Event Cycle Classes** worksheet, a portion of which is shown below.

	EVENT NAME	EVENT TYPE	SHIFT	TBD/TTD: Dist ID	: Progress	: Prob > 0	TTU: Di
2	Annual Maintenance	Distribution		TBD - Annual Maintenar	Time	1	TTU - A
3	Bi-Annual Maintenance	Distribution		TBD - Bi-Annual Mainte	Time	1	TTU - Bi
4	Brake Cycle	Distribution		TTD - Failure Brake	Time	1	TTU - Re
5	Drivetrain Chain Cycle	Distribution		TTD - Failure Drivetrain	Time	1	TTU - Re

- Copy all the cells from **rows 2 - 10** and **columns A - I only**. DO NOT copy the headers or row numbers.

Open the Event Cycles database table

- In ExtendSim, open the dialog of the *Event Builder* block
- At the bottom of the dialog, click the **Edit Classes Table** button; this opens the Event Cycles Classes table of the RBD database as shown below.



ECCP Record Index[1]	***Event Name[2]***	Event Type[3]	Shift[4]	TBD/TTD: Dist ID[5]	TBD/TTD: Progress Type[6]	TBD/TTD: Prob > 0[7]	TTU: t ID[8]

- Give the Database > Append New Records command or click the **Append New Records** button in the table's toolbar
- In the Append Records dialog, enter the number **9** and click **OK**; this results in a table that can hold 9 records

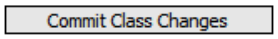
Paste the definitions into the Event Builder

⚠ Notice that the Event Cycle Classes table's first column (ECCP Record Index) is pink. Do not paste any data into that first column as it is reserved for internal use.

- In the database table, click in the cell of the **second column (Event Name)**, at **row 1**
- Give the **Paste/Paste Cells** command, causing the definitions to be copied into that table
- You should get a message reminding you to click the Commit Class Changes button after the definitions have been copied into the Event Builder.
- If you have enough room on the monitor, leave the database table open to see the changes as you go through the next steps. Otherwise, you can close the table.

ECCP Record Index[1]	*Event Name[2]*	Event Type[3]	Shift[4]	TBD/TTD: Dist ID[5]
0	*Annual Maintenance*	Distribution		TBD - Annual Maintenance
0	*Bi-Annual Maintenance*	Distribution		TBD - Bi-Annual Maintenance
0	*Brake Cycle*	Distribution		TTD - Failure Brake

- At the bottom of the Event Builder's dialog, click the **Commit Class Changes** button. This saves the data to the RBD's other data structures.



- At the right of the *Event cycle classes table* field, click the **Open** button to verify that the ECCP Record Index field (the pink column on the left) has been written to such that each record in the ECCP field contains a number.
- Close the Event Cycles database table.

⚠ Use the Commit button anytime you add records to the RBD database. If you don't click the Commit button, the ECCP Record Index won't get updated, causing database errors.

- Close the Event Builder's dialog
- Save your model

The event cycles classes will now be available for use as instances by the Start and Component nodes.

Associate the event cycles with the nodes

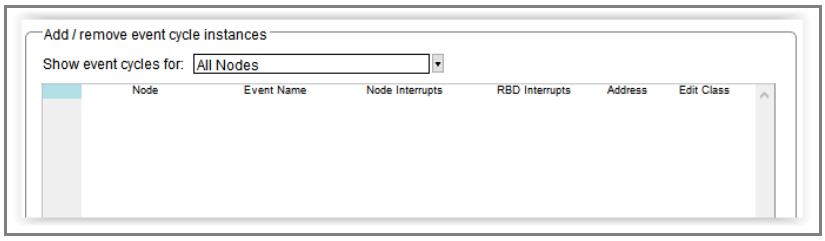
The only thing left to do is to specify which class of event cycles each node will use during the simulation. An event cycle class can have multiple instances, such as the Wheel Failure used by both the Front Brake and the Rear Brake. And each node can use more than one event cycle; for example, the Drivetrain has five event cycles—one for each subcomponent of the drivetrain.

You can associate an event cycle with a Component either directly using the Component's dialog or using the Start Node's dialog. This example uses the Start Node to add event cycle instances for the Start Node as well as for the Components.

⚠ If an event cycle will take the entire RBD down, it should be specified as a Start Node event.

Open the Start Node

- Open the dialog of the **Start Node**
- Select the block's **Event Cycles** tab
- Go to the **Add/remove event cycle instances** frame, shown below



Bicycle maintenance

The bicycle RBD has two event cycles that will cause the bicycle to be unavailable for a period of time, no matter what else is happening:

1) The annual maintenance on the drivetrain.
2) The bi-annual maintenance of the front and rear wheels.

Note that the drivetrain and front/rear event cycles will happen at their appointed times regardless of any down events for other parts of the bicycle. And since both of these will affect the availability of the bicycle, the event cycles for these maintenance events will be associated with the Start Node.

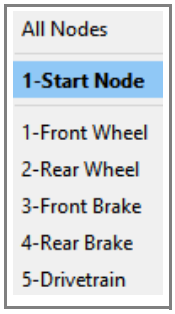
✦ The maintenance of the wheels and drivetrain impacts when their next down events will occur. For example, once the drivetrain is serviced, the event cycles for the drivetrain subcomponents need to be reset to use new TTD's. How this is accomplished will be shown in "Event cycle induced interrupts" on page 39.

Drivetrain's annual maintenance

The event cycle for the drivetrain is named Annual Maintenance. On page 33 you imported this event, along with others, into the table in the Event Builder. Now you need to associate it with the RBD.

▸ In the Start Node's *Add/remove event cycle instances* frame:

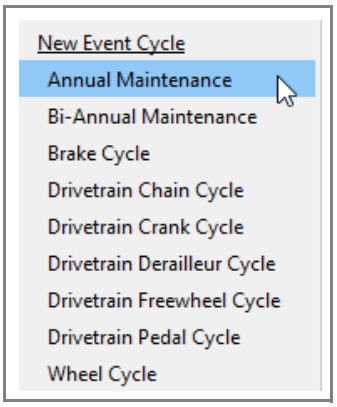
▸ Choose to **Show event cycles for: Start Node**



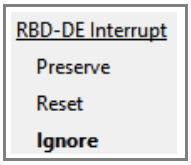
At this point, there are no event cycles, so the field is empty.

▸ At the bottom of the *Add/remove event cycle instances* frame, click the **Add** button

- In the popup menu, select **Annual Maintenance**



- For the message that appears (RBD-DE Interrupt) select **Ignore**. With this choice, the RBD going down will not affect the timing of the Annual Maintenance.



✦ As will be discussed in "Associate the interrupts" on page 38, the interrupt settings define if, when, and how a specific event cycle can be interrupted if the entire RBD (due to an RBD down event) or a node in the RBD (due to a component down event) goes down. The bicycle's maintenance events are going to occur independent of RBD or node state changes, so you should choose to ignore any interrupts.

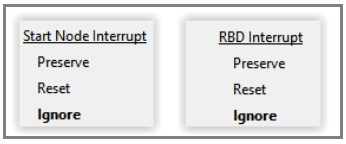
The list of event cycle instances for the Start Node should now look like this:

	Node	Event Name	Node Interrupts	RBD Interrupts	Address
1	1-Start	Annual Maintenance	Ignore	Ignore	1:12:0:1

Wheels' bi-annual maintenance

Both wheels get checked twice a year. Their event cycle is named Bi-Annual Maintenance.

- Repeat the above steps, adding **Bi-Annual Maintenance** as a Start Node event cycle.
- For the two interrupt messages that appear (Start Node Interrupt and RBD Interrupt) select **Ignore**.



✦ As was true for the drivetrain's annual maintenance, the wheels will be checked twice a year independent of node or RBD state changes. So those interrupts are ignored.

Start Node Down Event Interrupt (SN-DE Interrupt)

Notice that only one "interrupt" choice—the RBD-DE Interrupt—appeared when you entered the event cycle named Annual Maintenance. Yet when you added a second event cycle, there were two interrupt choices—one for the RBD and one for the Start Node. Also note that the word "Undefined" now appears in the SN-DE Interrupts column for Annual Maintenance, as shown below.

	Node	Event Name	SN-DE Interrupts	RBD-DE Interrupts	Address
1	1-Start	Annual Maintenance	Undefined	Ignore	2:11:0:1
2	1-Start	Bi-Annual Maintenance	Ignore	Ignore	2:11:0:2

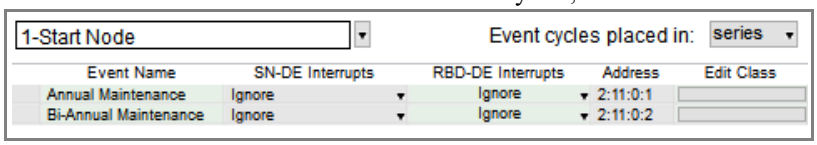
✦ If a component has only one event cycle, there are only two things that will cause that component to have a down event: that event cycle or something else that causes the entire RBD to go

down. However, if a component has more than one event cycle, there are additional possibilities for a down event because those other event cycles could affect each other.

When a component has more than one event cycle, you need to specify what effect that component's other event cycle instances will have. In this case, since the Bi-Annual Maintenance will have no effect on the Annual Maintenance, so you should choose "ignore".

- Click on the popup arrow in the SN-DE Interrupts column for the Annual Maintenance, and change the setting from Undecided to **Ignore**. (If you forget to do this, ExtendSim will remind you when you close the block's dialog.)

The Start Node should now have 2 instances of event cycles, as shown below.

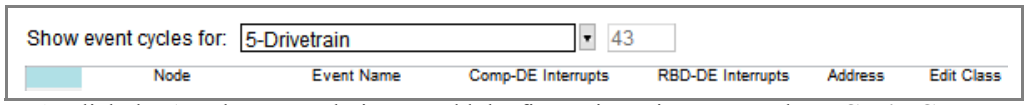


✦ Notice that, since the event cycles are placed in *series* rather than in parallel, the Start Node's icon on the model worksheet is stacked to the right. Being in series means that when *either* of those event cycles are in a down state, the Start Node (and hence the entire RBD) will go down. If the two event cycles were to be placed in parallel, *both* would have to be in a down state for the RBD to go down.

Drivetrain non-maintenance event cycles

You've associated the Drivetrain's annual maintenance with the Start Node. However, the Drivetrain also has 5 subcomponents—Chain, Crank, Dérailleur, Freewheel, and Pedals—and each subcomponent has its own event cycle. Those event cycles need to be associated with the Drivetrain. As before, you can do that from the Start Node's dialog.

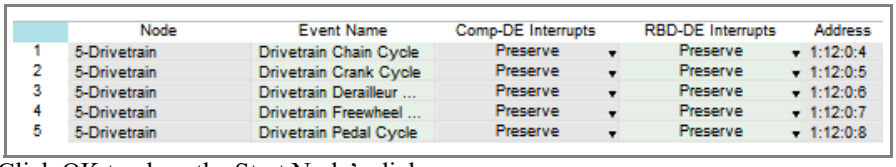
- In the Start Node's *Add/remove event cycle instances* frame, choose to **Show event cycles for: Drivetrain**



- Click the **Add** button each time to add the five Drivetrain event cycles—**Chain Cycle, Crank Cycle, Dérailleur Cycle, Freewheel Cycle, and Pedal Cycle**—to the Drivetrain.
- For each Comp-DE Interrupt message, select **Preserve**
- For each RBD-DE Interrupt message, select **Preserve**
- Since the Drivetrain has more than one event cycle, for whichever drivetrain subcomponent you added first, change the Comp-DE Interrupt from *Undefined* to **Preserve**.

✦ Some of the event cycles need to keep track of their progress towards their next down events, even if the RBD goes down. Choosing Preserve means that, when the RBD comes back up or if the drivetrain goes down because another event cycle in the drivetrain had a down event, those event cycles will pick up where they left off.

The event cycle instances for the Drivetrain should look as shown here:



	Node	Event Name	Comp-DE Interrupts	RBD-DE Interrupts	Address
1	5-Drivetrain	Drivetrain Chain Cycle	Preserve	Preserve	1:12:0:4
2	5-Drivetrain	Drivetrain Crank Cycle	Preserve	Preserve	1:12:0:5
3	5-Drivetrain	Drivetrain Derailleur ...	Preserve	Preserve	1:12:0:6
4	5-Drivetrain	Drivetrain Freewheel ...	Preserve	Preserve	1:12:0:7
5	5-Drivetrain	Drivetrain Pedal Cycle	Preserve	Preserve	1:12:0:8

- Click OK to close the Start Node's dialog.
- Save the model to save your changes.

Notice that the Drivetrain icon is stacked to the right, indicating that its associated event cycles are in series. If even one of those event cycles has a down event, the Drivetrain will go down. And since there is no redundancy or load sharing for the Drivetrain, the entire RBD will also go down.

✦ The same event cycles shown for the Drivetrain in the Start Node will be displayed in the Drivetrain's Event Cycles tab.

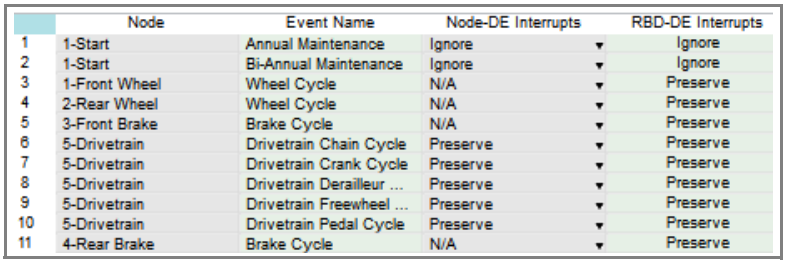
Brakes

- In the Start Node's *Add/remove event cycle instances* frame, choose to **Show event cycles for: Front Brake**
- Click the **Add** button and add the **Brake Cycle**
- For both the Component-DE Interrupt and the RBD-DE Interrupt popups, select **Preserve**
- Duplicate the above steps to add the **Brake Cycle** to the **Rear Brake**

Wheels

- In the Start Node's *Add/remove event cycle instances* frame, choose to **Show event cycles for: Front Wheel**
- Click the **Add** button and add the **Wheel Cycle**
- For both the Component-DE Interrupts and the RBD-DE Interrupts popups, select **Preserve**
- Duplicate the above steps to add the **Wheel Cycle** to the **Rear Wheel**
- Save the model

If you choose in the Start Node to show event cycles for all nodes, you should see:



	Node	Event Name	Node-DE Interrupts	RBD-DE Interrupts
1	1-Start	Annual Maintenance	Ignore	Ignore
2	1-Start	Bi-Annual Maintenance	Ignore	Ignore
3	1-Front Wheel	Wheel Cycle	N/A	Preserve
4	2-Rear Wheel	Wheel Cycle	N/A	Preserve
5	3-Front Brake	Brake Cycle	N/A	Preserve
6	5-Drivetrain	Drivetrain Chain Cycle	Preserve	Preserve
7	5-Drivetrain	Drivetrain Crank Cycle	Preserve	Preserve
8	5-Drivetrain	Drivetrain Derailleur ...	Preserve	Preserve
9	5-Drivetrain	Drivetrain Freewheel ...	Preserve	Preserve
10	5-Drivetrain	Drivetrain Pedal Cycle	Preserve	Preserve
11	4-Rear Brake	Brake Cycle	N/A	Preserve

Associate the interrupts

An event cycle's normal down event scheduling can be interrupted when:

1) The RBD goes down

2) A Component goes down

3) An event cycle in a Component or the Start Node affects another event cycle in that node

The options you selected for the interrupt settings in "Drivetrain's annual maintenance" on page 35, (SN-DE Interrupt and RBD-DE Interrupt) determine what happens if the RBD or the Start Node goes down. Those events are ignored, since they would have no effect on when the next annual maintenance would occur.

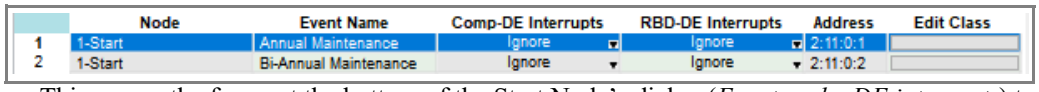
You might expect, however, that servicing the bicycle's front wheel would cause that wheel's next down event to be postponed. That is an example of one event cycle affecting the timing of another event cycle.

Event cycle induced interrupts

The occurrence of a down in one event cycle can interrupt the normal scheduling of other event cycles' down events. In the bicycle RBD, the down events from the drivetrain's annual maintenance and for the wheels' bi-annual maintenance result in the drivetrain and wheel event cycles needing to be "reset" to use a fresh set of TBD's or TTD's.

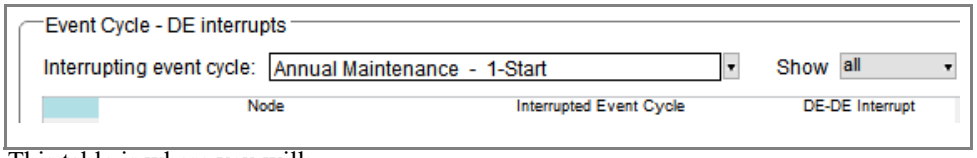
A table in the Start node is used to specify when and how these types of interrupts affect specific event cycles.

- In the Start Node, go to the Event Cycles tab
- Choose to show event cycles for the **Start Node**
- Select the first row in the cycles table by clicking row heading #1, as shown below



	Node	Event Name	Comp-DE Interrupts	RBD-DE Interrupts	Address	Edit Class
1	1-Start	Annual Maintenance	Ignore	Ignore	2:11:0:1	
2	1-Start	Bi-Annual Maintenance	Ignore	Ignore	2:11:0:2	

This causes the frame at the bottom of the Start Node's dialog (*Event cycle–DE interrupts*) to display the table below. Notice that the interrupter is the Annual Maintenance.

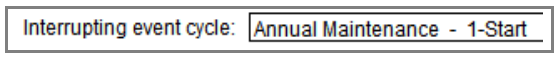


This table is where you will:

- Choose which event cycles will be affected (interrupted) by the occurrence of a maintenance down event
- Specify how and when the interrupt will affect the event cycle

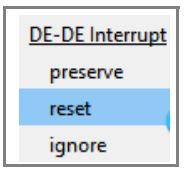
Annual maintenance and the Drivetrain Chain Cycle

With the Annual Maintenance event cycle selected as the interrupter:



- In the *Event cycle DE interrupts* table, click the **Add** button

- From the popup menu that appears, select the **Drivetrain Chain Cycle**
- In the DE-DE Interrupt popup that appears, select **reset**



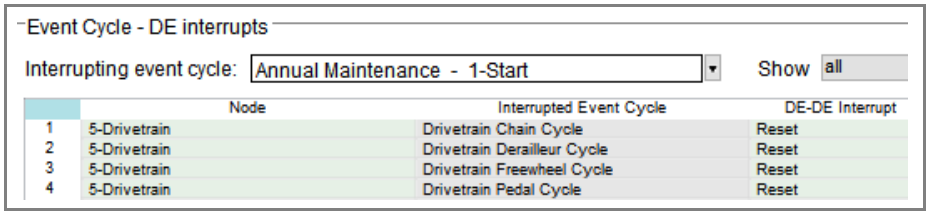
Other event cycles affected by the annual maintenance

As mentioned in the assumptions, the Crank is not serviced as part of the annual maintenance. However, the other subcomponents are serviced, which affects their next down events.

- **Skipping the Crank**, repeat the above steps for the remaining 3 drivetrain subcomponents:
  - Drivetrain **Dérailleur** Cycle
  - Drivetrain **Freewheel** Cycle
  - Drivetrain **Pedal** Cycle
- Save your model

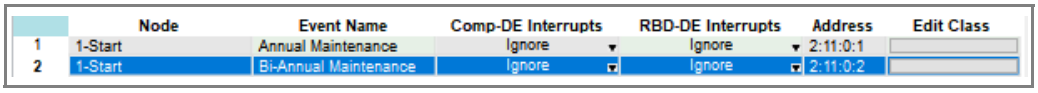
⚠ Since it cannot be serviced as part of the annual maintenance, do not add the Drivetrain Crank Cycle to the table. For Tutorial 2 you will simulate what happens when the crank needs repair.

The interrupts should appear as below:



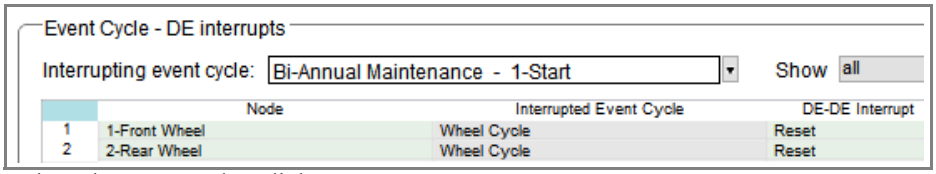
Bi-annual maintenance and the wheels

- Select the second row in the cycles table by clicking row heading #2, as shown below



- In the *Event cycle induced interrupts* table, click the **Add** button
- From the popup menu that appears, select **Wheel Cycle - Front Wheel**
- From the popups, choose **reset** for the DE-DE Interrupt.
- Repeat the above steps to add **Wheel Cycle - Rear Wheel**

The table should appear as below:



- Close the Start Node's dialog
- Save your model

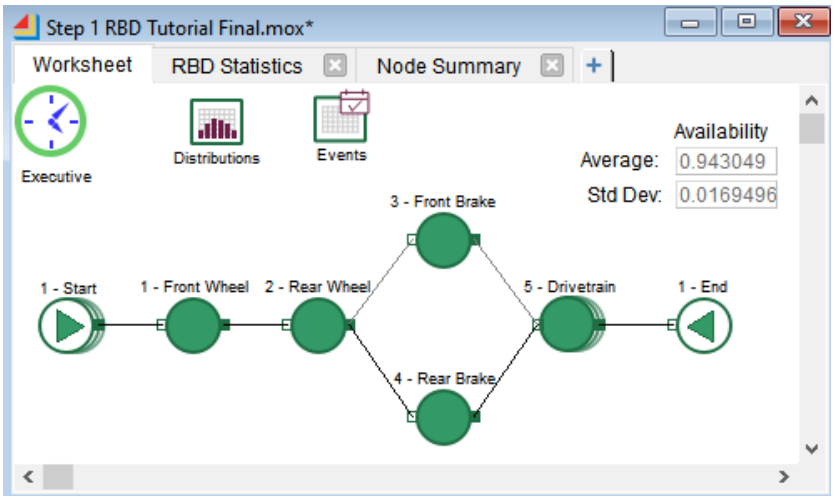
Availability

At this point you've entered all the information needed for the bicycle RBD. So that your model looks more like the example model on page 10, clone the availability field from the Results tab of the Start Node onto the model worksheet. This is the average availability of the entire RBD as calculated for the 5 runs.

Conclusion

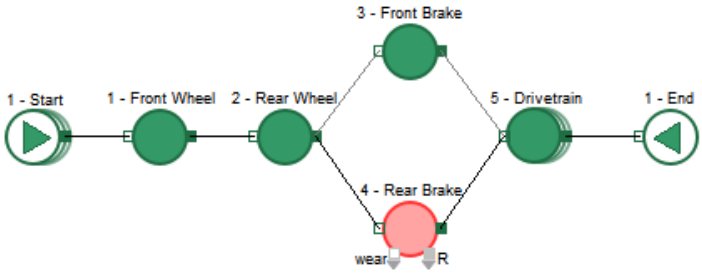
Your creation of the RBD is complete.

If you've followed all the steps, your model should be similar to the Step 1 RBD Tutorial Final model shown below and located at Documents/ExtendSim/Examples/Tutorials/Reliability.



Run the model

Even though it runs five times for a simulated period of three years, the model may run too fast for you to see changes. You can slow the model speed considerably by turning on animation and setting animation speed at the slowest. This allows you to see when the nodes are up or down and when the RBD is up or down.



Results

As mentioned in the Introduction, an RBD graphically and statistically describes when scheduled and unscheduled downs occur for individual resources and what impact that has on the availability of entire system. You see the graphical description when you run the simulation with animation on. The statistical description is reported in the Start Node.

Overall RBD results section

- Double-click the icon of the Start Node to open its dialog
- Go to its **Results** tab.

✦ When a node's dialog is opened, its icon turns yellow; it stays yellow until the dialog is closed.

The *Overall RBD results frame* at the top of the Results tab will be similar to what is shown below. Since this model uses random numbers for its distributions, your results will differ.

Overall RBD results

	Down Events	Total Downs	Up Time	Down Time	Availability
Average:	58.2	38	1031.8458	63.154228	0.9423249
Std Dev:	2.4819347	1.8973666	13.621606	13.621606	0.0124398

The simulation is run five times and the results for each run are shown in the table in the RBD statistics frame. The statistics at the top of the frame report the average for all the runs.

Notice that the sum of the Up Time and Down Time for each of the runs is 1095. This corresponds to the model's simulation time of 1095 days.

Database section

The Start Node's Results tab also has a *Database* frame where you can access various reports.

Database

	Results Tables	Create	Open	Collect Data	Overlapping Events Are	For Overlapping Events, Pick
1	RBD Summary	☑		all runs	independent	N/A
2	Node Summary	☑		last run	independent	N/A
3	Event Summary	☐		last run	independent	N/A
4	Event Log	☐		last run	independent	N/A

Each subsequent table in the frame collects finer and finer detail about what happened in the RBD. For example, the RBD Summary table is set to collect data for all runs and provides a high-level picture of what happened to the RBD, while the Event Log records every event during the last run or all runs.

The Node Summary table provides information about the Start Node and the five Components in this RBD. As selected in the Collect Data column's popup menu, the information is for the last run (row 5 in the RBD Statistics table) rather than for all runs. (Simulation runs are numbered starting at 0, so the last run is run number 4.)

	Node[1]	Down Events[2]	Total Downs[3]	Up Time[4]	Down Time[5]	Availability[6]	Run Number[7]
1	1-Start	9	6	1090.500000	4.500000	0.995890	4
2	1-Front Wheel	0	0	1095.000000	0.000000	1.000000	4
3	2-Rear Wheel	1	1	1092.128016	2.871984	0.997373	4
4	3-Front Brake	12	12	1010.071622	84.928378	0.920123	4
5	5-Drivetrain	23	23	1076.444242	18.555758	0.982796	4
6	4-Rear Brake	12	12	1011.724644	83.275356	0.920680	4

The Event Summary table gives high level metrics for each individual event cycle associated with the RBD. The Event Log provides a chronological record of every event that occurred during each run.

Things to notice from the reports in the Database frame

✦ Your numbers will be slightly different, but the concepts are the same.

- The RBD Summary table gives information about the entire RBD, while the Node Summary table gives information about each node.
- The Down Events for each run in the RBD Summary table will be the same as the sum of the Down Events for that run in the Node Summary frame (57 for the last run, as shown above).
- However, as seen for the last run in the two windows, the sum of the actual Total Downs for the RBD (40) is less than the Total Downs for all the nodes (54). This makes sense due to the redundancy of the brakes; the RBD doesn't go down unless both of the brakes are down so there are fewer downs for the RBD itself than for the nodes.
- When there are downs for the Front and Rear Wheels it is due to failures, not to their bi-annual maintenance. Since the entire bicycle is brought off line anytime maintenance is done, the event cycles for bi-annual maintenance have been associated with the Start Node rather than with the wheels.
- The simulation runs for 3 years. Each year there is an annual maintenance and two bi-annual maintenances, resulting in three maintenance events per year. However, since the second bi-annual maintenance occurs at the same time as the annual maintenance, there are only 2 Total Downs per year associated with the Start Node, for a total of 6.

Next steps

The next chapter shows how to integrate the RBD you just built with a discrete event model.

Reliability
Tutorial & Reference

Tutorial 2: Adding PSS to RBD

The previous chapter showed how to create a stand-alone RBD. The purpose of this chapter is to demonstrate how to integrate an RBD with a model built using the ExtendSim process simulation software (PSS). In particular, it shows how the RBD you built in the previous tutorial can become more representational of the real system if it is combined with an item-based process model of the message delivery business.

⚠ If you aren't already familiar with the ExtendSim Item library, see the Discrete Event QSG (Quick Start Guide) located at Documents/ExtendSim/Documentation.

What PSS integration adds to the model

Adding process simulation to RBD leads to a better understanding of the impact that availability has on overall system performance. In the bicycle example, integration provides more information than the availability number provides, such as how changes in availability impact the business's ability to deliver messages.

In addition, the occurrence of down events in the stand-alone RBD must by definition be based merely on the progression of time, since no other relevant factor is available. However, when an RBD is integrated with process simulation, the process model can supply the RBD with usage information that reflects the wearing of the components. This allows for a more realistic representation of the process, showing that how much a component is used affects when its next down event will occur.

✦ Integrating process simulation with RBD allows the RBD's components to progress to a down event based on factors other than the mere advance of time.

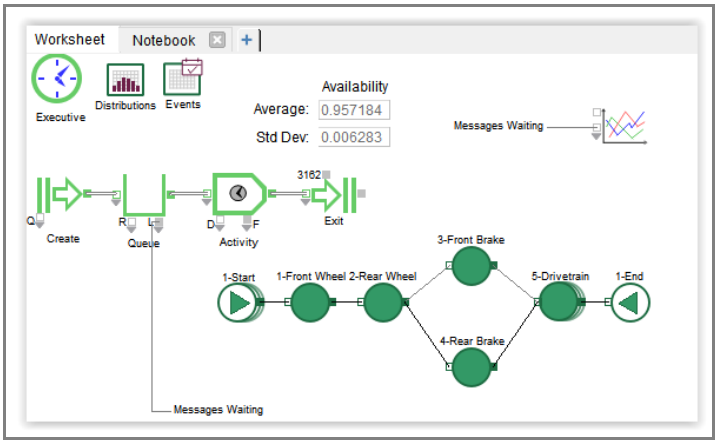
The tutorial models in this chapter

There are two parts to this tutorial:

1) Integrate an item-based process model (2A) with the RBD you created in the previous tutorial.
2) Explore what happens if the wait for delivering the messages gets too long and messages renege (2B).

✦ So that you can focus on the integration of RBD with PSS, both sections of the tutorial start with a pre-built model that includes process simulation and an RBD.

The 2A RBD Tutorial model



▸ Open the model *Step 2A RBD Tutorial* located at Documents/ExtendSim/Examples/Tutorials/Reliability.)

✦ Don't open the "Final" model.

▸ Save the model as MyRBDItems

What the model does

This model adds the RBD you built in the previous tutorial to a simple MM1 queuing model that represents the message delivery process.

- In the process section of the model, messages that need to be delivered arrive randomly from a Create block, sit in a Queue while they wait to be delivered, and are processed by an Activity block. The Activity represents the bicycle's operation and it tells the RBD whether the bicycle is being used or not.
- The RBD determines the bicycle's availability (its up and down states) based on the wearing of the Components. It reports that information to the Activity block, letting it know if and when the bicycle is available to deliver messages.
- At the end of the simulation, the down events and the messages waiting for delivery are displayed on the graph.

Assumptions for the process portion

So that you can focus on the integration, the process portion of the model has already been completed. As you can see in the dialogs of the Create, Queue, and Activity blocks:

- Messages arrive randomly at the Create block, represented by an exponential distribution with a mean of 0.35 days
- They are stacked in a first-in-first-out (FIFO) order as they wait in the Queue for delivery
- The bicycle messenger can only deliver one message at a time. In the Activity block, the delivery time is specified by a triangular distribution with the following parameters:
  - Minimum: 0.10 days
  - Maximum: 0.50 days

- Most likely 0.25 days

The RBD portion

The annual and bi-annual maintenance events are calendar-based and will occur at set times no matter what. Thus the following TBD/TTD event cycles are the only ones that would be affected by wearing:

- Brake Cycle
- Drivetrain subcomponents: Chain, Crank, Dérailleur, Freewheel, and Pedal cycles
- Wheel Cycle

As seen in the Event Builder's Event Cycle Classes table, progress towards the next down event for these components is time.

Brake Cycle	Distribution		TTD - Failure Brake	Time
Drivetrain Chain Cycle	Distribution		TTD - Failure Drivetrain Chain	Time
Drivetrain Crank Cycle	Distribution		TTD - Failure Drivetrain Crank	Time
Drivetrain Derailleur Cycle	Distribution		TTD - Failure Drivetrain Derailleur	Time
Drivetrain Freewheel Cycle	Distribution		TTD - Failure Drivetrain Freewheel	Time
Drivetrain Pedal Cycle	Distribution		TTD - Failure Drivetrain Pedal	Time
Wheel Cycle	Distribution		TTD - Failure Wheel	Time

✦ Wearing can affect progress towards the next down event and thus the duration of the time-between-downs (TBD) or time-to-down (TTD). However, it never impacts the time to up (TTU). Thus only Progress Types that are TBD/TTD need to be changed.

Integrating the two portions

Your task for this tutorial is to:

1) Change the TBD/TTD progress type for the affected event cycles from *time* to *wear*
2) Connect the RBD and process simulation sections so they can exchange data with each other

Change the event cycle progress type

There are two ways you can change the TBD/TTD Progress Type from *time* to *wearing*:

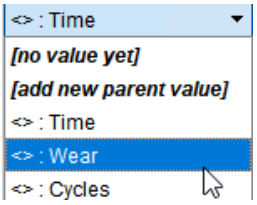
1) Modify an event cycle in the Event Builder's dialog. In the *Time between downs (TBD)/Time to down (TTD)* frame, change the event cycle's Progress from "Time" to "Usage", select Wear as the Usage type, and save the change. Repeat for each event cycle and save the model.



2) Or, open the Event Cycle Classes table in the dialog of the Event Builder, make all the changes there, and save your model. This is the approach you will use for this tutorial.

▸ In the dialog of the Event Builder, open the Event Cycle Classes table. Notice that, in the TBD/TTD Progress Type column, all event cycles are set to Time, as shown in the screen-shot above.

- In the *TBD/TTD Progress Type* column, for the **Brake Cycle**:
  - Use the popup menu to change its progress type from **Time** to **Wear** as shown here
  - After the cell has switched from Time to Wear, copy the word **Wear** as it is displayed in the cell
- For the Drivetrain Chain Cycle, paste into the Progress Type cell so that Time is replaced by **Wear**.



✦ Although copy/paste is fast, you could alternatively use the popup to select Wear for each event cycle.

- Paste **Wear** into the Progress Type cells for the five remaining event cycles:
  - Drivetrain Crank Cycle
  - Drivetrain Dérailleur Cycle
  - Drivetrain Freewheel Cycle
  - Drivetrain Pedal Cycle
  - Wheel Cycle

The table should now show all the event cycles, except for the Annual and Bi-Annual Maintenance, having a progress type of Wear.

TBD/TTD: Dist ID[5]	TBD/TTD: Progress Type[6]
TTD - Annual Maintenance	Time
TTD - Bi-Annual Maintenance	Time
TTD - Failure Brake	Wear
TTD - Failure Drivetrain Chain	Wear
TTD - Failure Drivetrain Crank	Wear
TTD - Failure Drivetrain Derailleur	Wear
TTD - Failure Drivetrain Freewheel	Wear
TTD - Failure Drivetrain Pedal	Wear
TTD - Failure Wheel	Wear

- Close the database table, click OK in the Event Builder's dialog to close it, and save the model to save the changes you've made to the database.

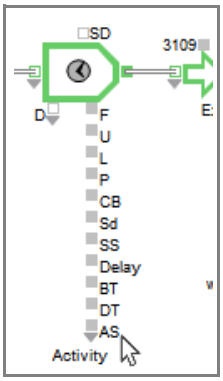
✦ Since you haven't added any records, you don't need to Commit Event Cycle Class Changes, you just have to save the model.

Connect the process model to the RBD

The easiest way for the RBD and the process model to exchange data is by drawing connection lines between the blocks.

Activity

- On the Activity block's Shutdown tab:
  - Check the box to **Enable shutdown**.
  - Leave the other options as they are and click OK to close the dialog and save changes.
- Notice that there is now an SD (shutdown) input connector at the top of the Activity block's dialog. This will allow the RBD to control when the Activity (the bicycle) is able to deliver messages and when it is not.
- At the bottom of the Activity block's icon, drag down on the variable output connector to reveal the AS (Advanced Status) output as shown here.



End Node

- In the End Node's *Connectors* tab, choose to **Show output connectors**, then click OK to save and close the dialog. This puts a variable output connector at the bottom of the End Node's icon.
- Drag down on the variable connector until the *up* output connector is revealed.
- Draw a connection line (or make a named connection) between the **down (RBD)** output of the End Node and the **SD** input on the Activity

Start Node

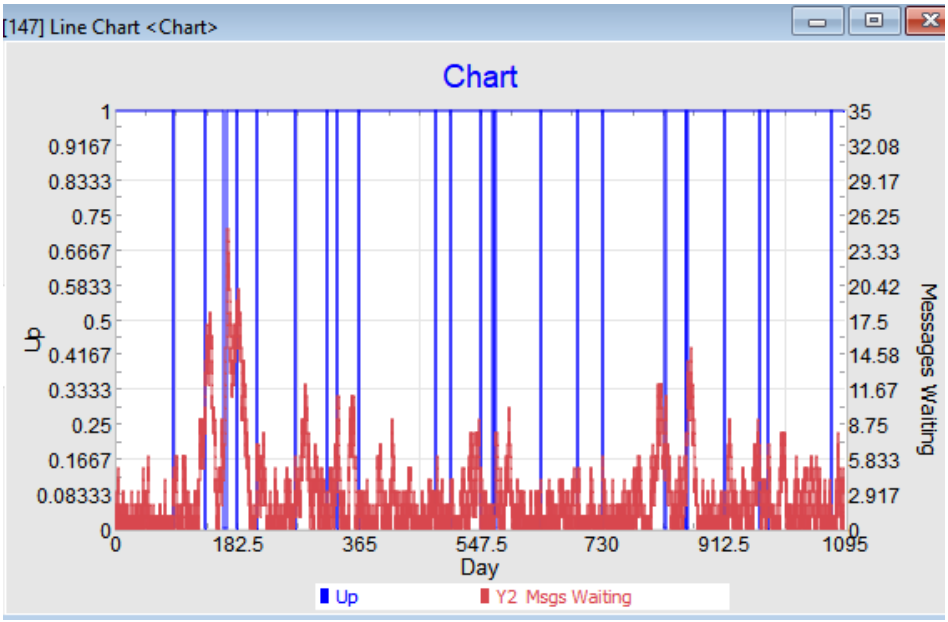
- In the Start Node's *Connectors* tab, select to **Show input connectors**, then close the dialog.
- Notice that this puts a variable input connector at the bottom of the Start Node's icon and the first input is **wear**.
- Draw a connection line from the Start Node's **wear** input to the **AS** output on the Activity.

Make named connections

- Create a named connection named Up and use it in 2 places:
  - Connected to the *up* output of the End Node
  - Connected to the top input on the Chart block
- Save the model to save your changes

Run the simulation

The graph should appear similar to the one shown below. This shows the affect that the bicycle's unavailability has on the buildup of messages in the Queue.



Reliability
Tutorial & Reference

Tutorial 3: Add Reliability to a Rate Model

The previous chapter showed how to integrate an RBD with a discrete event ExtendSim model. In this chapter, we'll introduce reliability-driven equipment availability into an existing discrete rate process model.



If you aren't already familiar with the ExtendSim Rate library, see the Discrete Rate QSG (Quick Start Guide) located at Documents/ExtendSim/Documentation.

Overview

In many real-world systems, equipment is not continuously available due to failures, repairs, and maintenance activities. Adding an RBD to a Rate model allows these reliability characteristics to be incorporated directly into the process simulation. As equipment transitions between operational and failed states, the process model responds accordingly, providing a more accurate assessment of system availability, throughput, and operational performance.

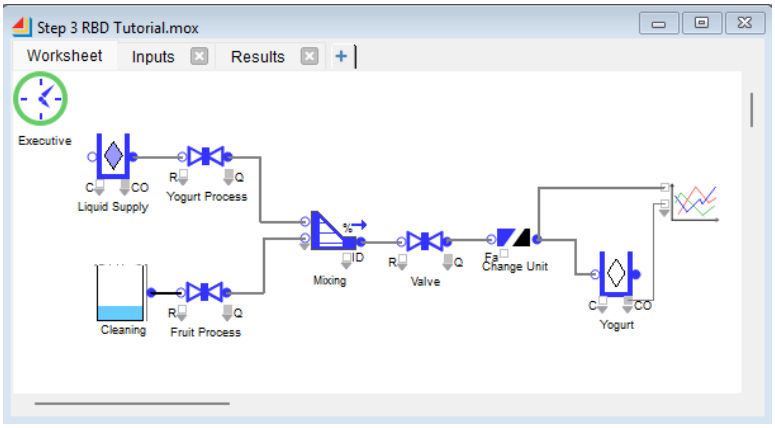
Why use Reliability with Rate models?

Adding reliability to a Rate model enables analysts to:

- Quantify the impact of equipment failures on production.
- Identify bottlenecks caused by downtime.
- Compare alternative maintenance strategies.
- Evaluate design changes that improve system availability.
- Estimate realistic throughput rather than assuming perfect equipment performance.

This integration is particularly valuable when modeling manufacturing systems, processing plants, maintenance-intensive operations, or any environment where equipment failures influence overall productivity.

The Step 3 RBD Tutorial model



- Open the model Step 3 RBD Tutorial.mox located in the Documents/ExtendSim/Examples/Tutorials/Reliability folder.
- ✦ Don't open the "Final" model.
- Save the model as MyRBDRate.

What the model does

This model represents a simplified yogurt production process in which upstream material streams are blended in a mixer and then sent to a packaging or changeover unit before becoming finished yogurt. Rather than assuming the mixer is always available, we're going to add a Reliability Block Diagram (RBD) to calculate the operational state of the mixer. When the mixer is available ("Up"), production proceeds normally. When the mixer fails ("Down"), material flow through the process will be interrupted until the mixer is repaired.

Adding the RBD will demonstrate how equipment failures and repairs affect system performance allowing a traditional process-flow model to account for real-world equipment behavior and providing a more realistic estimate of system availability and production capacity.

Rate model structure

The rate portion of the model consists of the following blocks from the Rate library representing specific activities in the yogurt product process:

Tanks - Liquid supply source at the beginning of the process and a tank at the end of the process to store the yogurt

Valves – Yogurt and fruit process activity plus another Valve to transition the materials from the mixing process to final production

Merge - Mixing operation

Change Units – To convert the flow from gallons to containers

Plus, a hierarchical block consisting of blocks from the Item library to represent a Cleaning operation

From Rate Model to Reliability-Driven Process Model

Let's add an RBD that controls whether the mixer is available to process material. The RBD will determine the mixer's operational state and communicate that state to the process model.

The RBD will evaluate the operational state of the mixer throughout the simulation. Failure and repair behavior are defined for the mixer using parameters such as failure distributions, repair distributions, or event-cycle definitions created with the Event Builder.

The RBD will produce a state output that indicates whether the mixer is:

- Up (operational)
- Down (failed)

Step 1: Add Reliability blocks

From the Reliability library, add the following blocks to the process model:

- **Start Node**
- **Component**
- **End Node**

These blocks define a simple reliability system consisting of a single critical component.

Step 2: Build and connect the RBD

Connect the blocks as follows:

Start Node → Component → End Node

Start Node

- Open the dialog of the Start Node block. In the General tab, enter Start in the Name field.
- In the Start Node's Connectors tab, select to Show input connectors, then close the dialog.
- Notice that this puts a variable input connector at the bottom of the Start Node's icon and the third input is sched (schedule).
- Draw a connection line from the Start Node's sched input to the AS output on the Valve.

Component

- Open the dialog of the Component block. In the General tab, enter Mixer in the Name field.

End Node

- Open the dialog of the End Node block. In the General tab, enter End in the Name field.
- In the End Node's Connectors tab, choose to Show output connectors, then click OK to save and close the dialog. This puts a variable output connector at the bottom of the End Node's icon.
- Drag down on the variable connector until the down (CS) output connector is revealed.
- Draw a connection line (or make a named connection) between the down (CS) output of the End Node and the SD input on the Valve.

Add another Line Chart (Chart library) to the model

- In the End Node, drag down on the variable connector until the up output connector is revealed.
- Draw a connection line (or make a named connection) between the up output of the End Node and the input on the Line Chart.

Step 3: Define Mixer failure and repair behavior

We will define three event cycles and then assign them to the Mixer node. Two event cycles are for randomly occurring events – one for maintenance and one for mixer failure – and one will be for when supply has dwindled and the valve is down because it is starved of supply:

- Open the Event Builder block dialog and go to the Event Cycles tab
- Create the Maintenance event cycle:
  - Click on the drop-down menu by Event cycle name and choose New Event Cycle Class.
  - Name the event Maintenance.
  - For Event cycle type, choose Distribution.
  - For RBD Interrupt, choose Ignore.
  - Create the Time between downs (TBD) distribution:
    - •Make sure TBD is selected for Type.
    - •Click on the Distribution drop-down under TBD and choose New TBD Distribution.
    - •Name the distribution Maintenance.
    - •In the Distribution Builder dialog that opens:
      1) Make sure that the Maintenance TBD distribution is selected.
      2) Select TBD for the Distribution purpose.
      3) Choose the Constant distribution.
      4) Set the constant value to be 1, time units to be months.
      5) Click the red Save button.
      6) Click OK to close the dialog.
  - Create the Time to Up (TTU) distribution:
    - •Click on the Distribution drop-down under TTU and choose New TTU Distribution.
    - •Name the distribution Mixer.
    - •In the Distribution Builder dialog that opens:
      1) Make sure that the Mixer Failure TTU distribution is selected.
      2) Select TTU for the Distribution purpose.
      3) Choose the Normal distribution.
      4) Set the mean value to be 3, std dev to 0.5, time units to be hours.
      5) Click the red Save button.
      6) Click OK to close the dialog.
  - In the Event Builder dialog, click the red Save button.
- Create the **Block Starve event cycle**:
  - Click on the drop-down menu by Event cycle name and choose New Event Cycle Class.
  - Name the event Block Starve.

    - For Event cycle type, choose sched connector.
    - In the Event Builder dialog, click the red Save button.
- Close the Event Builder dialog.
- Double-click the Start Node to open its dialog and select the Event Cycles tab.
- Set up the system-wide event cycle for the Start Node:
    - Select Start Node from the Show event cycles drop-down menu.
    - Click the Add button and choose Block Starve.
    - Start Node interrupts should be Ignore, and RBD interrupts should also be Ignore.
- Set up the Mixer events:
- Select Mixer from the Show event cycles drop-down menu
    - Click the Add button and choose Maintenance.
    - For RBD Interrupts, choose Ignore.
    - Click the Add button and choose Mixer Failure.
    - For Component interrupts, choose Preserve.
    - For RBD Interrupts choose Preserve.
    - Now that there are two event cycles for the Mixer, go the Maintenance row and select Ignore for Component interrupts.
- Click OK to close the Start Node dialog.

Step 4: Report Mixer availability on the worksheet

Availability is calculated as a fraction of simulation time that the mixer remains operational:

$$Availability = \frac{\text{Uptime}}{\text{Uptime + Downtime}}$$

To view the overall availability of the system, open the **Start Node** block dialog and select the *Results* tab. If the simulation is configured to execute multiple runs, then the average and standard deviation for uptime, downtime, and availability will be reported. If only a single run is executed, then the *average* will be the value of each for that run, and the standard deviation will be zero.

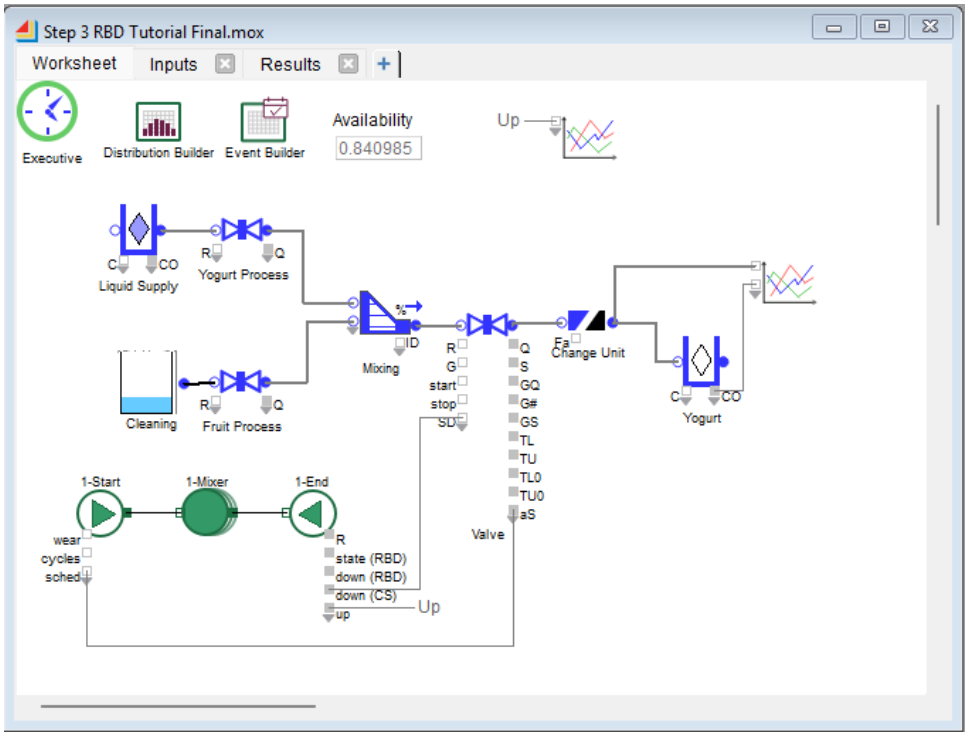
Step 5: Run the Simulation

Run the model and observe the results.

- When the Mixer reliability component is Up, material continues to flow through the process.
- When the Mixer reliability component is Down, processing stops until a repair returns the mixer to service.
- Material flow resumes automatically when the Mixer is repaired.

Results

To compare your model to ours, see the model named Step 3 RBD Tutorial Final.mox in the folder Documents/ ExtendSim/Examples/Tutorials/Reliability.



The final model demonstrates how reliability affects system performance. Equipment failures and repairs now influence production throughput, system availability, and overall process behavior. By connecting the RBD state output to the Mixing block's Up input, the model realistically represents the impact of mixer downtime on yogurt production.

Key Takeaway

This model demonstrates one of the most common reliability modeling techniques in ExtendSim: using a Reliability Block Diagram to determine the operational state of a critical resource and linking that state directly to a process block. This allows equipment failures and repairs to dynamically control system behavior, providing a more realistic representation of production performance and availability.

Reliability
Tutorial & Reference

Tutorial 4: Adding a Gantt Chart to an RBD Using Python

⚠ **If you aren't already familiar with the ExtendSim Rate library, see the Discrete Rate QSG (Quick Start Guide) located at Documents/ExtendSim/Documentation.**

Overview

The goal of this tutorial is to add a Gantt chart to a reliability model, enabling the visualization of down-times experienced by individual components as well as the entire system. While there is not currently a native Gantt chart block, the Python Bridge block allows ExtendSim to leverage additional capabilities in the Python ecosystem. We will be using the Plotly library to create a simple Gantt chart, along with the Python Bridge block's database access capabilities and persistent Python object storage to create the chart.

Pre-load Python libraries

This tutorial requires that the Python environment used by ExtendSim be pre-loaded with the pandas and plotly libraries. These can be installed using pip:

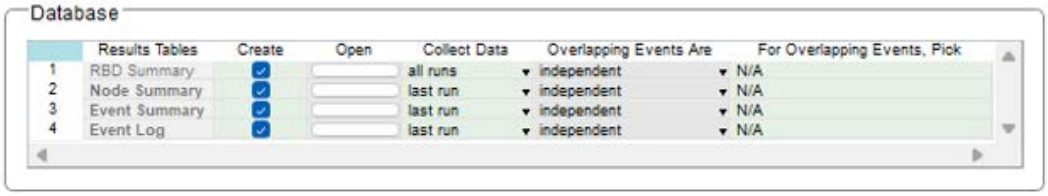
```
pip install pandas

pip install plotly
```

Adding the Gantt Chart to an existing model

This tutorial builds on Step 1 of the RBD Tutorial, so load the model you built from **Tutorial 1: Creating an RDB**. Or use the pre-built final model at Examples\Tutorials\Reliability\Step 1 RBD Tutorial Final.mox.

Open the dialog of the **Start Node**. In the Results tab, confirm **Event Log** will be generated, as shown below:



Adding blocks to the model

Place an Equation block (Value library) into the model as shown in the figure below. Configure the block to use one output connector and no input connectors. The equation should be as follows:

```
outCon0 = EDGetStartDate();
```

- Where *outCon0* is the name of the single output variable, which may be renamed as desired.
- The *EDGetStartDate* function returns the start date of the simulation as a number in the Excel date format; this will be passed to the Python scripts to set the time axis of the Gantt chart.

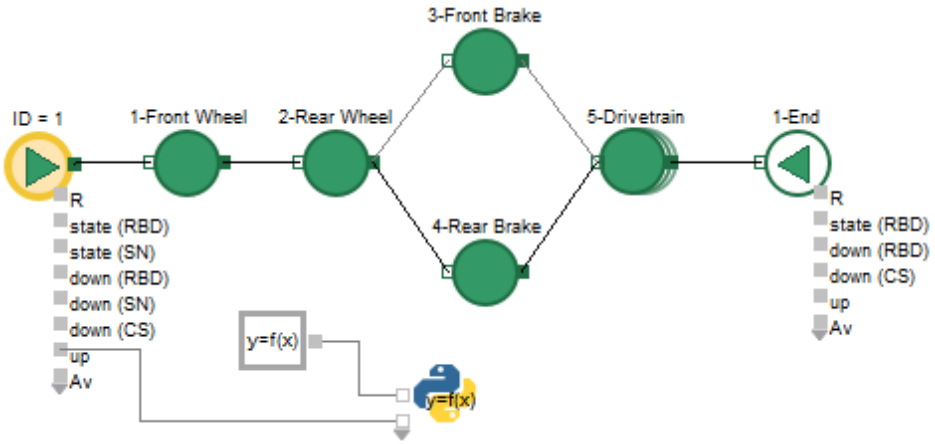
The start date can be set from the *Run* > *Simulation* Setup menu option; set it to Use Calendar dates and to start on January 1, 2021. Note that the pre-configured run time is 1095 days, which is three years; thus, the end date should be January 1, 2024.

Open the Equation block's *Options* tab. Check "When receive output value connector msgs", leaving everything else unchecked. It should respond to any message at any output value connector and should respond immediately. This will ensure that the block does not send values to the Python Bridge block and potentially trigger the execution of the Python code unnecessarily.

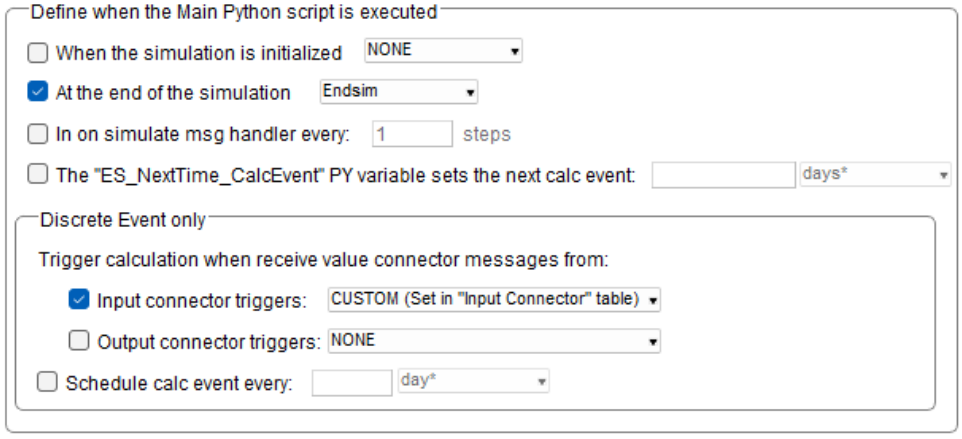
Python Bridge block

Most of the data that's going to be plotted in the Gantt chart is stored in the database - the log tables that say when each component is up or down. What those tables don't store, though, is whether or not the whole system is up or down. For example, when one brake is down, the bike can still go, but when both brakes are down, then the whole bike can't be ridden. That system-level up/down information comes from the **Start Node**'s up connector, and is collected and stored in the Python Bridge block. At the end of the simulation, it will take that information from the individual component log entries, and plot the Gantt chart.

Place a **Python Bridge** block (Value library) into the model and connect it to the **Equation** and **Start Node** blocks as shown in the figure below. Note that the variable output connectors on the Start Node block must be expanded to reveal the "up" connector, and the variable input connectors on the Python Bridge block must be expanded to two connectors.



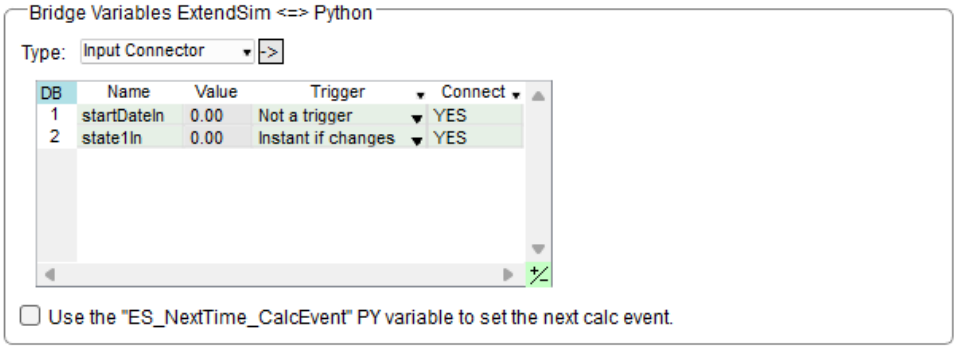
Open the dialog of the **Python Bridge** block. In the **Options** tab, the execution conditions should be configured as shown below. By setting the Input connector triggers to "CUSTOM," it enables the selection of different behaviors for each input; these will be configured in the next step. Whenever the state of the system changes, as reported by the connected **Start Node**, the state will be recorded by the Python code. Additionally, the Python code will collect data from the event log stored in the database and populate the Gantt chart accordingly. Thus, the script also needs to be run at the end of the simulation, so it is configured to run on *EndSim*.



Go to the *Python Code* tab. Configure the input variables as shown in the image below.

The first input connector, which is connected to the **Equation** block that sends the simulation start date, should be named startDateIn and should not trigger execution of the script; its value will be pulled when the script is initiated in other ways.

The second input, which is connected to the *up* value output of the **Start Node** block, should be named *state1In*. This will trigger the execution of the Python code only if the value has changed. There are no other conditions that should trigger script execution in this model.



Configuring the Initialization Script

Open the Initialization Editor by clicking the button labeled Create Script (Init). Note, once the script is created, this button will be labeled Access Editor (Init). Insert the following code segments (only the code in Courier New typeface) into the script, in order:

- Package imports:

```
import pandas as pd

import plotly.express as px

from datetime import datetime, timedelta
```

- Date conversion function definition, which converts Excel dates to Python dates. Note the *timeUnitsPerDay* constant, which should be consistent with the model's time units:

```
timeUnitsPerDay = 1.0

def convertDate(excelDate):

    baseDate = datetime(1899, 12, 30)

    return baseDate + timedelta(days=excelDate)
```

- Database constants, which identify the name of the database and the name of the event log table(s). A single table can be stored in *tableName*. Multiple tables can be stored in the *tableNameList* array:

```
dbName = "RBD"

# tableNameList = ["Results: Event Log_XX", "Results: Event Log_YY",
"Results: Event Log_ZZ"]

tableNameList = [ ]

# tableName = "Results: Event Log_XX"

tableName = "Results: Event Log_1"
```

- Variable initialization, which helps track the state of the overall system and record state change events in *dataList*:

```
lastState = -1;

downTimeStart = -1;

dataList = []
```

- Database index setup, which remembers the numeric indices of database tables for easier access when executing the main script. Note that the field names may be different based on the version of the Reliability library being used:

```
dbIndex = es.DBDatabaseGetIndex(dbName)

if tableName == "" and len(tableNameList) > 0:

    tableName = tableNameList[0]

tableIndex = es.DBTableGetIndex(dbIndex, tableName)

componentFieldName = "Node"

componentFieldIndex = es.DBFieldGetIndex(dbIndex, tableIndex, com-
ponentFieldName)

eventFieldName = "Event Name"

eventFieldIndex = es.DBFieldGetIndex(dbIndex, tableIndex, event-
FieldName)

startTimeFieldName = "DE Start Time"

startTimeFieldIndex = es.DBFieldGetIndex(dbIndex, tableIndex,
startTimeFieldName)

durationFieldName = "TTU"

durationFieldIndex = es.DBFieldGetIndex(dbIndex, tableIndex, dura-
tionFieldName)
```

- Optional notification to confirm setup success:

```
es.UserNotify(f"component: {componentFieldIndex}, event: {event-
FieldIndex}, startTime: {startTimeFieldIndex}, duration: {dura-
tionFieldIndex}", 0)
```

Configuring the Main Script

Next, open the Main Editor by clicking the button labeled *Create Script (Main)*. Note, once the script is created, this button will be labeled *Access Editor (Main)*. Insert the following code segments (only the code in Courier New typeface) into the script, in order. Be careful to maintain the indentation on each line, as it is significant to Python. Note that some long lines may wrap in this document, though a copy-and-paste should preserve indentation correctly:

- Restrict execution to the last run of a series, otherwise a new graph will be created for each run (or, if this is desired, remove this statement):

```
# Only collect and plot data for the last run

if es.gvars.CurrentSim() == es.gvars.NumSims() - 1:
```

- During the simulation, this code notes changes in up/down system state as reported by the **Start Node** block, and stores an entry in the *dataList* whenever the system recovers from a downtime event:

```
    # Executing during simulation

    if es.gvars.CurrentTime() < es.gvars.EndTime():

      # If the RBD was down and is now back up, record the down event

        if state1In == 1 and lastState == 0:

            dataList.append(dict(Component="RBD",

                                 Event="RBD Down",

                                 Start=convertDate(startDateIn +
downTimeStart),

                                 Finish=convertDate(startDateIn +
es.gvars.CurrentTime())))

            downTimeStart = -1;

        # Or, if the RBD is now down but wasn't before, note when
the downtime started

        elif state1In == 0 and lastState != 0:

            downTimeStart = es.gvars.CurrentTime()

# After processing, the current state becomes the last state

        lastState = state1In
```

- At the end of the simulation, translate downtime events from the system log table(s) into the *dataList*:

```
    # Finalization phase

    else:

        if len(tableNameList) == 0:

            tableNameList.append(tableName)

        numRowsCopied = 0

        for currentTableName in tableNameList:

          tableIndex = es.DBTableGetIndex(dbIndex, currentTableName)

            es.UserNotify(f"Processing table {currentTableName},
index {tableIndex}", 0)
```

```
for currentRow in range(1, es.DBRecordsGetNum(dbIndex, tableIndex)
+ 1):

              component = es.DBDataGetAsString(dbIndex, tableIndex,
componentFieldIndex, currentRow)

                if (component != ""):

                  event = es.DBDataGetAsString(dbIndex, tableIndex,
eventFieldIndex, currentRow)

                  startTime = es.DBDataGetAsNumber(dbIndex, tableIn-
dex, startTimeFieldIndex, currentRow)

               duration = es.DBDataGetAsNumber(dbIndex, tableIndex,
durationFieldIndex, currentRow)

                    start = convertDate(startDateIn + (startTime /
timeUnitsPerDay))

                    finish = convertDate(startDateIn + ((startTime
+ duration) / timeUnitsPerDay))

                    newRow=dict(Component=component, Event=event,
Start=start, Finish=finish)

                    dataList.append(newRow)

                    numRowsCopied = numRowsCopied + 1
```

- Once all events have been transferred to the *dataList*, generate the graph. Note that in some cases, this code may be executed on *FinalCalc* prior to population of the event log tables. The check on the number of rows copied should prevent generating a graph too early, and should allow the graph to be generated on *EndSim*:

```
        #Only plot if we copied rows from the log; otherwise we may
be too early, e.g. in FinalCalc

         if numRowsCopied > 0:

             df = pd.DataFrame(dataList)

             fig = px.timeline(df, x_start="Start", x_end="Finish",
y="Component", color="Event")

           fig.update_yaxes(autorange="reversed") # otherwise tasks
are listed from the bottom up

             print(f"Showing for data frame shape {df.shape}")

             fig.show()
```

Results

To compare your model to ours, see the model named *Step 1 RBD Tutorial Final – Python Gantt Chart.mox* in the folder Documents/ ExtendSim/Examples/Tutorials/Reliability.

Once the simulation runs, the plotly library will direct the browser to load the Gantt chart on the *fig.show()* command. There will be controls at the top right of the window that allow you to zoom, filter, and manipulate the graph to explore results as desired. Each downtime will show some data about each event when the mouse is over the event on the graph.

Reliability Tutorial & Reference

Reference

RBD terminology

Term	Definition
Availability	Percentage of time a resource is in an “up” state and available to perform work.
Component	The nodes located between a Start Node and an End Node in the interior of an RBD. Components represent resources and are placed in series and/or parallel to each other. Over time, Components alternate between up and down states due to event cycles.
Distribution	A description of a random phenomenon that specifies the length of time a resource will be in an up or a down state.
Down Event	A Down event is the point in time when an event cycle switches from the up state to the down state. Down events are either scheduled or unscheduled and can be due to failure, off-shifting, maintenance, or repair.
Edges	Edges describe how nodes in an RBD are related to each other. In ExtendSim, edges are represented by the connection lines between nodes.
Event Cycle	Describes an alternating behavior of cycling through up and down states over time. Event cycles are sometimes called Failure modes, although that is a more limiting term because not all downs are caused by failures.
Failure Mode	The manner in which a design, process, product, or service will cycle through failure and non-failure states over time.
k of N	The “k” defines the minimum number of upstream parallel components (N) that need to be in an Up state in order for the “one downstream component” to be up.
Load Sharing	A form of redundancy in which there is a parallel structure that supports the workload. If one of the load sharing components fails, it causes a higher share of the workload for the remaining components, increasing the wear rate.
Nodes	The interconnected shapes that form the structure of the RBD. They consist of a Start Node, an End Node, and one or more Components.
Parallel Nodes	A set of Component nodes placed in parallel to each other indicates redundancy. Unless the entire set of parallel nodes is down, that section of the RBD will be available to perform work.

Term	Definition
PSS	Process simulation software. Event-based tools such as the ExtendSim Item library for discrete event simulation or Rate library for discrete rate simulation.
RBD	Reliability block diagram. A network of interconnected nodes that alternate over time between up and down states in order to model the availability of the system as a whole. A model may have more than one RBD; each RBD has a Start Node, an End Node, and one or more Components.
Reliability	The probability the RBD will remain in its up state for the entire duration of the run.
Resources	The means by which process activities and operations are performed. Their lack of availability can cause constraints on the system.
Scheduled Downs	A planned Down event such as for maintenance or off-shifting.
Serial Nodes	A set of Components that are placed one after the other. The entire set must be up for that part of the RBD to be available to perform work.
Standby	The case in which a component has a backup component in a Down (idle) state until it is needed. Note that the backup component could have the same failure rate while down as it does while up, causing it to be a “hot standby”.
TBD	Time-Between-Downs. The period of time between the start of a Down event and the start of the following Down event.
TTD	Time-To-Down. The period of time between the start of an Up event and the start of the following Down event.
TTU	Time-To-Up. The period of time between a Down event and the following Up event.
Unscheduled Downs	An unexpected Down event such as a failure.
Up Event	An Up event is the point in time when the event cycle switches from the down state to the up state.

For a pictorial representation of many of these terms, see “Event cycles” on page 68.

✦ By default ExtendSim assumes the system is in an up state at the start of the simulation.

Example models

✦ The following models are located at Documents/ExtendSim/Examples/Reliability/RBD Bicycle. They are variations on the RBD model of the bicycle discussed in the tutorial chapters.

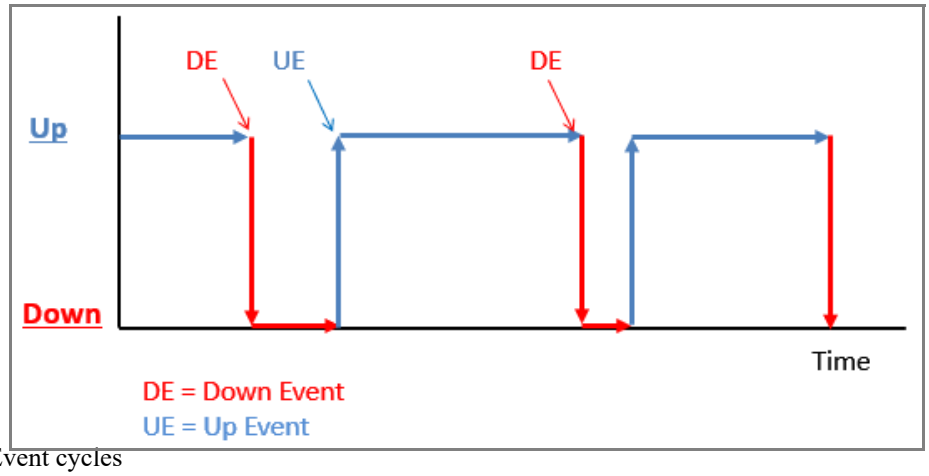
Model	Description
Shift Blocks	Same as the Step 1 RBD model discussed in Tutorial 1 except it uses Shift blocks (Item library) to specify both the annual and bi-annual maintenance event cycles.
Item-Based Repair	Demonstrates how to translate repair events into items that must travel through a repair process before returning to the RBD in a fixed state. By default, repairs are modeled using a random distribution to define how long the repair will take. However, that black boxing of the repair process might not provide a high enough level of fidelity. When the availability of the critical resources needed to make repairs affects repair duration in meaningful ways, Reliability users have the option to use the ExtendSim process modeling capabilities to model the repair process with a higher level of fidelity.
Standby (A)	Uses an Equation block (Value library) that is wired directly to the RBD to model "Standby". Standby is a general purpose reliability concept where a component sits idle until it is called into service. In the model the rear brake is not used until the front brake fails.
Standby (B)	The same as the Standby "A" model except the Equation block is not wired to the RBD with connection lines. Instead the ExtendSim internal database and Link Alerts notify the Equation when the front or rear brake has gone into the failed state. The Equation then issues the Standby directive (discussed above) to the RBD via the database instead of via connection lines.
Load Share (A)	Load share is another general reliability concept where two or more components (in this case the front and rear brakes) share the load. When one of the components goes down, the wear rate on the other component(s) speeds up. This model uses an Equation directly wired to the RBD.
Load Share (B)	The same as the Load Share "A" model but wireless. as discussed above for the Standby (B) model.
k of N (A)	The "k of N" is a general reliability concept used to define how many (k) of the upstream parallel components (N) need to be in an up state in order for the "one downstream component" to be up. In this model, k = 1, the upstream parallel components are the front and rear brakes, and the "one downstream component" is the Drivetrain. That is, the drivetrain will be considered "up" if at least 1 of the 2 brakes is up. This model uses an Equation block wired directly to the RBD to determine how many of the parallel components are up and whether the "one downstream component" should be up or down.
k of N (B)	The same as the k of N "A" model but wireless, as discussed above for the Standby (B) model.

Basics

Up events and Down events

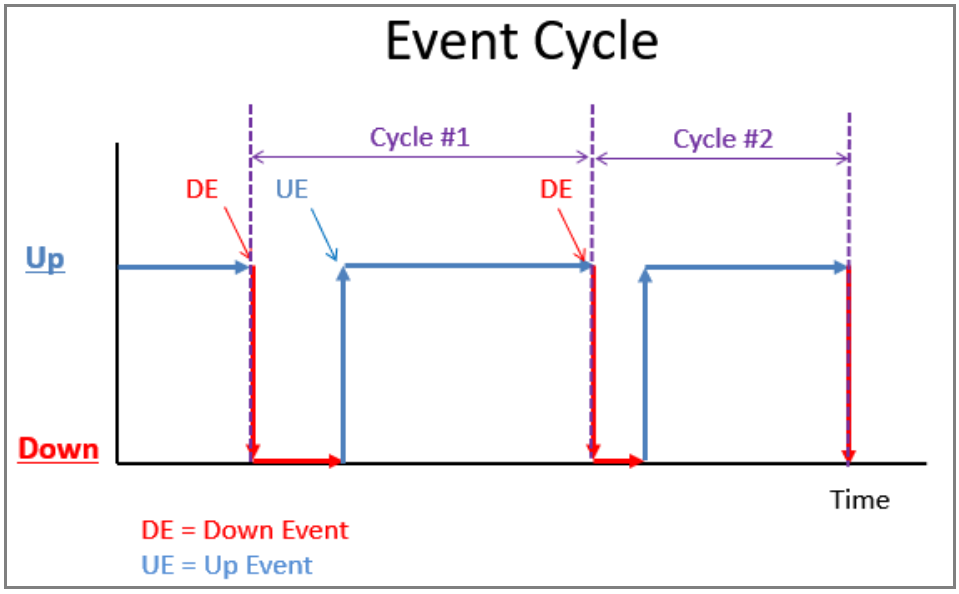
An Up event (UE) is the point in time when a resource becomes available. By default, ExtendSim assumes that all resources are available at the start of the simulation.

A Down event (DE) is the point in time when a resource becomes unavailable. Down events are either scheduled or unscheduled and can be due to failure, off-shifting, maintenance, or repair.



Event cycles

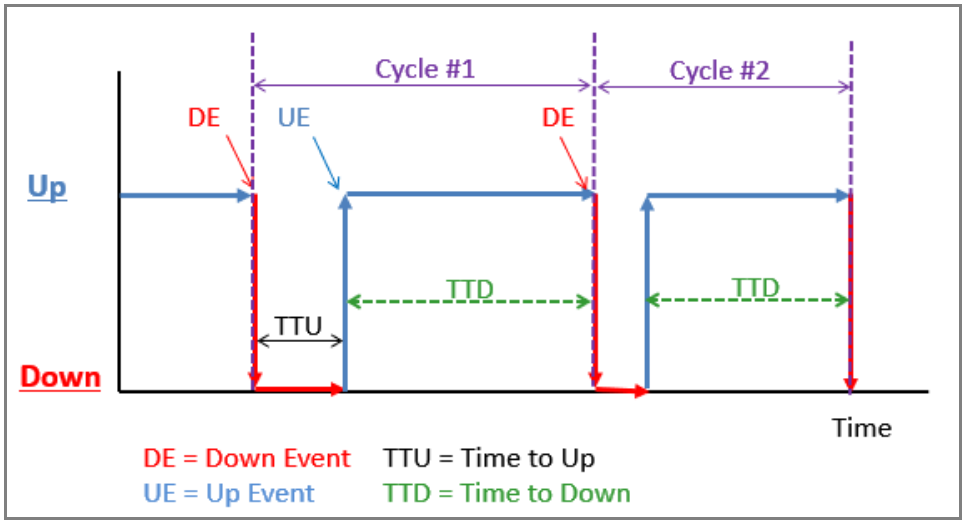
An event cycle describes how a resource alternately cycles through its up and down states over time. As shown below, this event cycle has multiple up and down cycles.



TTU's and TTD's

As seen in the graphic below:

- There is a period of down time that starts with a Down event (DE) and ends with an Up event (UE). The time between those two events is determined by a distribution that specifies the Time-to-Up (TTU).
- There is a period of up time that starts with an Up event and ends with a Down event. The time between these two events is determined by a distribution that specifies the Time-to-Down (TTD).


Cycle #1
Cycle #2
DE
UE
DE
Up
TTD
TTD
TTU
Down
Time
DE = Down Event
TTU = Time to Up
UE = Up Event
TTD = Time to Down

Index

A-C

Add/remove event cycle instances frame 38
availability 21
  definition 2
Bicycle model 10
Components 6
  adding to the RBD 25
  defined 16
components
  in parallel 3
  in series 2

D-F

Database frame 42
DE-DE Interrupt popup 40
definition of reliability block diagram 2
Distribution Builder 6
  Creating a distribution class 27
  defined 19
down events 2
edges
  definition 2
End Node 6
  defined 16
Event Builder 6
  defined 17
Event cycle induced interrupts 39
Event Cycles
  Creating an event cycle class using the Event Builder 31
  Event cycle classes 30
  Event cycle table 30
event cycles
  definition 3, 6, 17
  in series 37
Event Log 43
Event Summary table 43
example models 8
fail modes
  definition 3
failures 2

G-I

How To chapters 8
Ignore 36
Interrupts
  event cycle induced 39
interrupts 38
  ignore 36
  preserve 37
Item-Based Repair model 67

J-L

k of N 67
k of N (A) model 67
k of N (B) model 67
load share 67
Load Share (A) model 67
Load Share (B) model 67

M-N

model
  example 8
Node Summary table 42, 43
nodes
  definition 2

O-P

Overall RBD results frame 42
PFS 4
  integrated with RBD 4
Preserve 37
process flow simulation 4
process flow simulation (PFS)
  compared to RBD 3
Process Simulation
  Adding PSS to RBD 45
Python Bridge block 58, 59
  Configuring the Initialization Script 60
  Python Code 59

Q-S

Rate Models
  Add Reliability to a Rate Model 51
RBD
  compared to process flow simulation 3
  databases 5
  definition 2
  description 2
  integrated with PFS 4
  structure 13
  terminology 65
RBD Summary table 43
RBD-DE Interrupt 36, 39
redundancy 3, 11
Reliability module
  features 6
  framework 5
  when to use 5

reset 40
run parameters 24
Shift Blocks model 67
simulation
    parameters 24
Simulation Setup command 24
SN-DE Interrupt 36, 39
Standby (A) model 67
Standby (B) model 67
Standby directive 67
Start Node 6
    asks for Component names 24
    Database frame 42
    explained 15
    icons 15
    Overall RBD results frame 42
Start Node down event interrupt 36, 39
Step 1 RBD Tutorial Final model 41

T-V

User Reference
    How To chapters 8