User's Guide

Reliability Module

GoldSim
TECHNOLOGY GROUP

Partial funding for the GoldSim Reliability Module was provided by NASA.

GoldSim Version 14.0 (October 2021)

GoldSim Technology Group
255 S. King Street, Suite 800
Seattle, Washington 98104
USA

Visit us at our web site: www.goldsim.com
Email us at: software@goldsim.com

Contents

Chapter 1: Introduction

> **The major difference between a thing that might go wrong and a thing that cannot possibly go wrong is that when a thing that cannot possibly go wrong goes wrong, it usually turns out to be impossible to get at or repair.**
>
> **Douglas Adams, *Mostly Harmless***

Chapter Overview

GoldSim is a user-friendly, highly graphical, object-oriented program for carrying out dynamic, probabilistic simulations to support management and decision-making in engineering, science and business.

The GoldSim Reliability Module is a program extension to GoldSim which allows you to probabilistically simulate the reliability and performance of complex engineered systems over time. GoldSim provides the ability to model the interdependence of components through requirements and fault trees, as well as the capability to define multiple independent failure modes for each component. This facilitates both reliability modeling and risk analysis.

Each of the GoldSim modules has a separate User's Guide describing its capabilities and features. This document provides a complete description of the features and use of the GoldSim Reliability Module.

Note: This document only describes the Reliability Module, a program extension to the GoldSim simulation framework. In order to take full advantage of all of the powerful features of the Reliability Module, you will eventually need to become familiar with the various features and capabilities of the underlying GoldSim simulation framework. When necessary, these features and capabilities are mentioned and discussed briefly in the current document. They are described in detail in a separate document, the **GoldSim User's Guide**.

In this Chapter

This introductory chapter discusses the following topics:

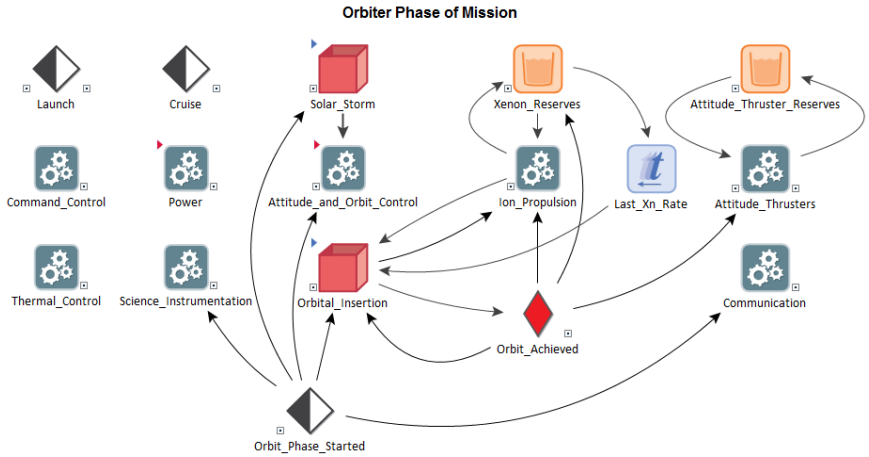
- What is GoldSim?
- What is the Reliability Module?
- How Does GoldSim's Approach Differ From Other Reliability Modeling and Risk Analysis Approaches?
- What is the Difference Between Reliability Professional and Reliability Learning Edition?

What is GoldSim?

GoldSim is a computer program for carrying out dynamic, probabilistic simulations.

As used here, *simulation* is defined as the process of creating a model (i.e., an abstract representation or facsimile) of an existing or proposed *system* (e.g., a business, a mine, a watershed, a forest, the organs in your body, the atmosphere) in order to identify and understand those factors which control the system and/or to predict (forecast) the future behavior of the system. Almost any system which can be quantitatively described using equations and/or rules can be simulated.

The GoldSim simulation environment is highly-graphical and completely object-oriented. That is, you create, document, and present models by creating and manipulating graphical objects representing the components of your system, data and relationships between the data:



In a sense, GoldSim is like a "visual spreadsheet" allowing you to visually create and manipulate data and equations. As can be seen in the example shown above, based on how the various objects in your model are related, GoldSim automatically indicates their influences and interdependencies by visually connecting them in an appropriate manner. GoldSim also sets up and solves the equations represented by the objects and their interdependencies.

The various objects with which a GoldSim model is constructed are referred to as ***elements***. Each element represents a building block of the model, and has a particular symbol or graphical image (which you can subsequently customize) by which it is represented on the screen.

Although the standard elements incorporated within GoldSim can be used to build powerful and complex models, it was realized from the outset of the development of GoldSim that specialized elements and features may be required in order to efficiently model some kinds of systems. As a result, GoldSim was designed to readily facilitate the incorporation of additional modules (program extensions) to enable the program to address specialized problems. The Reliability Module is one of these program extensions.

What is the Reliability Module?

Reliability engineering involves measuring and analyzing the ways that systems can fail (and be repaired) in order to increase their design life, and eliminate or reduce the likelihood of failures, downtime and safety risks. It involves developing a mathematical representation (a model) of an existing or proposed engineered system in order to predict the performance of the system over time. The system (e.g., a furnace) consists of multiple components (e.g., a blower, a burner). The output of these models typically consists of predictions of measures such as ***reliability*** (the probability that a component or system will perform its required function over a specified time period) and ***availability*** (the probability that a component or system is performing its required function at any given time). Reliability models are frequently used to compare design alternatives on the basis of metrics such as warranty and maintenance costs

For some systems, the analyst may be more concerned with ***risk analysis*** than with reliability. Risk analysis focuses on predicting the probability of those (presumably rare) failures that can lead to injury, loss of life, severe damage to the system, or perhaps damage to the surrounding environment. Hence, in a risk analysis, the output of the model typically is the probability of a particular unlikely, but high consequence outcome (e.g., catastrophic failure of the system), and identification of those events or components most likely to lead to that outcome. Risk analysis models are typically used to inform decisions about required levels of redundancy, and to evaluate system safety and risk.

The GoldSim Reliability Module is a program extension to GoldSim which allows you to probabilistically simulate the reliability and performance of complex engineered systems over time. The fundamental outputs produced by the Reliability Module consist of predicted reliability metrics (e.g., reliability and availability) for the overall system, and for individual components within that system. The Reliability Module can also be used to compute the probability of specific consequences (e.g., catastrophic failure of the system) to support risk analysis. GoldSim catalogs and analyzes failure scenarios, which allows for key sources of unreliability and risk to be identified.

How Does GoldSim's Approach Differ From Other Reliability Modeling and Risk Analysis Approaches?

When discussing how GoldSim differs from other approaches, it is useful to differentiate reliability modeling from risk analysis. The GoldSim Reliability Module can be used for both types of analysis. With conventional methods, however, these two types of analysis use very different types of tools (since they are focused on different types of results).

Conventional Approaches to Reliability Modeling

An excellent discussion of conventional approaches to reliability modeling is provided by Ebeling (1997), and readers who are not familiar with these approaches are encouraged to consult this text.

Most reliability modeling approaches involve the assumption of a static model, where the system configuration never changes (other than due to the failure/repair of components), and where its properties don't change with time. This is a convenient assumption, as it allows the use of simple techniques, such as closed form mathematical equations or reliability block diagrams. Markov chains are another conventional reliability approach, and although they introduce an element of dynamism, the system itself (and its properties) cannot change with time. Because of the simplifying assumptions required to use these conventional techniques, they may be inappropriate for some systems.

Some of the difficulties with using these approaches for complex systems are summarized below.

Closed-Form Equations. These methods are heavily dependent on classical models (i.e., they have been primarily developed for use with standard failure distributions like the Poisson and Weibull). Even if failure data can be fitted to a standard distribution, it is difficult to model complex systems with closed form equations. For example, if a system has two Weibull failure modes, they cannot be algebraically combined into a single Weibull failure mode for use with the Weibull reliability equation.

Reliability Block Diagrams. Reliability block diagram models are static and do not account for the highly dynamic nature of many systems. A reliability block diagram model also assumes the system is in steady state, and unless correction factors are used, assumes that all of its components are independent.

Markov Chains. Markov chains enumerate a number of system "states" and the probabilities for transitioning between these states. However, the number of transition probabilities (and the computational effort) required to solve a Markov chain grows exponentially with the number of states. Because of this "state-space explosion", in many cases a system must be greatly simplified in order to use a Markov chain approach.

Of course, the conventional approaches are appropriate for many systems, particularly when employed by an experienced practitioner. However, in some cases, a more realistic reliability model may be required.

Conventional Approaches to Risk Analysis for Engineered Systems

Risk analysis is a very broad field, utilizing a variety of quantitative approaches. In the current context, however, we are primarily concerned with risk analysis of complex engineered systems (e.g., nuclear power plants, infrastructure such as dams, and space and defense systems) that are composed of highly-reliable and frequently redundant components, which in most cases are required to have an extremely low risk of a catastrophic failure.

The conventional approach to risk analysis for such systems focuses on the analysis of initiating events and subsequent event sequences that could lead to failures, and on enumerating and calculating the probabilities of different outcomes through tree-based analytical procedures (event trees/fault trees). Stamatelatos et al, (2011) and Vesely et al. (2002) provide good descriptions of these approaches.

For many types of systems (e.g., nuclear power plant probabilistic risk assessments), these approaches work well. However, systems that are highly dynamic or have significant process variability can be very difficult to model realistically using event tree/fault tree approaches, and they require a tremendous amount of preprocessing effort.

As a result, an approach like GoldSim's that facilitates explicit representation of dynamics and variability potentially provides a powerful complement to existing methods.

The GoldSim Approach to Reliability Modeling and Risk Analysis

GoldSim is a general purpose dynamic, probabilistic (Monte Carlo) simulator. Dynamic simulation allows the analyst to develop a representation of the system whose reliability is to be determined, and then observe that system's performance over a specified period of time.

The primary advantages of dynamic probabilistic simulation are:

- The system can evolve into any feasible state and its properties can change suddenly or gradually as the simulation progresses.
- The system can be affected by random processes, which may be either internal (e.g., failure modes) or external.
- If some system properties are uncertain, the significance of those uncertainties can be determined.

In Monte Carlo simulation, the model is run many times with uncertain variables sampling different values each time (each run is called a ***realization***). These realizations are each considered equally likely (unless specialized sampling techniques are used), and can be combined to provide not only a mean, but also confidence bounds and a range on the performance of the system. In addition to the statistical data these realizations provide, multiple realizations may also reveal failure modes and scenarios that may not be apparent, even to experienced risk and reliability modelers.

In addition to providing a more accurate representation of uncertainty, GoldSim also allows you to create a more detailed and accurate representation of your system than can be achieved with even the most sophisticated risk and reliability methodology.

With GoldSim, you can:

Model Components that have Multiple Failure Modes: GoldSim allows you to create multiple failure modes for components, each of which can either be defined by a distribution or occur when a specified condition arises. Failures which occur according to a distribution do not have to use time as the control variable. For example, a vehicle might use mileage to define failure, while an aircraft might use the number of cycles.

Model Complex Interdependencies: In addition to providing a logic-tree mechanism to define relationships, GoldSim also allows you to model the more subtle effects of failure on other portions of the system. For example, you can easily model a situation where the failure of one component causes another component to wear more quickly.

Model the External Environment: Reliability elements in GoldSim are fully compatible with all other GoldSim elements. This means that the environment in which the system operates can also be modeled, and can affect and interact with the system.

These features and capabilities provide a powerful engine for realistically modeling the risk and reliability of complex engineered systems.

What is the Difference Between Reliability Professional and Reliability Learning Edition?

There are two versions of the Reliability (RL) Module: Reliability Professional and Reliability Learning Edition. Your copy of GoldSim will have one or the other (but not both), depending on your license. Reliability Learning Edition is included automatically with GoldSim Pro. Reliability Professional must be purchased separately.

Reliability Learning Edition provides all of the features and capabilities of Reliability Professional. The only difference between the two versions is that Reliability Learning Edition limits you to adding no more than ten reliability elements. Reliability Professional imposes no limits on the number of reliability elements that can be added.

Who Should Use the Reliability Module?

The GoldSim Reliability Module is intended for use by engineers, scientists, researchers and students who are interested in understanding and predicting the reliability and risk of complex systems, and the interaction of such systems with the outside world.

The software itself, although relatively complex, can be mastered by anyone familiar with the basic functions of a personal computer and the Windows operating system. The key requirements for applying the Reliability Module are a clear understanding of the physical system being modeled; and a basic understanding of uncertainty analysis and probability theory:

- Because the software was designed to be extremely flexible, it intentionally imposes few constraints on the inputs that you define. Hence, it is your responsibility to ensure that the system being defined is consistent and realistic. As a result, the most important requirement is that you have a clear understanding of the features, processes, failure modes and events controlling the behavior of the system to be modeled. This should include a good understanding of the fundamentals of reliability modeling and risk analysis.
- Although GoldSim can be run in a deterministic manner (i.e., with no specified uncertainty in input parameters), one of the key features of GoldSim is its ability to explicitly represent such uncertainty through the use of probability distributions. In order to do so, the user must have at least a basic understanding of the representation and propagation of uncertainty. Appendix A of the **GoldSim User's Guide** (a companion document to this manual, described below) provides a brief primer on this topic, along with suggestions for further reading.

How the GoldSim Documentation is Organized

The GoldSim Documentation Suite

The Reliability Module is a specialized extension to the basic GoldSim simulation framework. The document you are reading only describes the GoldSim Reliability Module, and assumes that you are somewhat familiar with the basic capabilities of GoldSim.

The basic capabilities of GoldSim are described in the **GoldSim User's Guide**. That document provides a complete description of the features and capabilities of the GoldSim simulation framework.

How this Manual is Organized

This document is organized into seven chapters and two appendices.

The seven chapters are as follows:

- **Chapter 1: Introduction**. The remainder of this chapter discusses the information required for you to get started using the GoldSim Reliability Module, including conventions used in the manual, activating the module, using online help, and obtaining technical support.
- **Chapter 2: Getting Started with the Reliability Module**. This chapter provides an overview of the features and capabilities of the Reliability Module, provides a brief overview of some key concepts of the GoldSim simulation framework, and walks you through a simple example in order to help you get started with the Reliability Module.
- **Chapter 3: The Reliability Elements**. This chapter describes the basic features of GoldSim's two reliability elements.
- **Chapter 4: Running a Reliability Simulation**. This chapter discusses how to set up and run both dynamic and static reliability simulations.
- **Chapter 5: Advanced Reliability Modeling Concepts**. This chapter describes some of the more advanced features of the reliability elements.
- **Chapter6: Displaying Reliability Results**. This chapter describes the reliability results available in GoldSim.
- **Chapter 7: Example Reliability Module Applications**. This chapter describes a number of examples which illustrate the application of the various features and capabilities of the Reliability Module.

The manual also includes two appendices:

- **Appendix A: Mathematical Details of the Reliability Module**. This appendix describes the mathematical details incorporated into the reliability elements.
- **Appendix B**: **Failure Mode Import Spreadsheet Format**. This appendix describes the format used when failure mode data is imported from a spreadsheet.

Learning to Use the Reliability Module

Although GoldSim's intuitive interface will tempt you to simply dive in and start playing with the software, you are strongly discouraged from doing so, even if you are an experienced modeler. Spending some time up front (by following the steps outlined below) is the quickest and most effective way to understand the software's features and capabilities and start building models using the GoldSim Reliability Module.

1. **Learn how to use the basic GoldSim framework first**. In order to use the Reliability Module, you must first have a basic understanding of the GoldSim framework. You cannot learn the extension without first learning the basic concepts underlying framework. At a minimum,

you should take the GoldSim Tutorial. The Tutorial is available from the GoldSim splash screen, and can also be accessed from the main GoldSim menu (**Help | Tutorial…**).

In addition to the Tutorial, a free "hands-on" online training Course (titled "Introduction to GoldSim") is available that will provide you with a thorough understanding of the key concepts on which GoldSim is based and all of the fundamentals required to build complex models of nearly any kind of system. (Note, however, that the Course only discusses basic GoldSim and does not discuss the Reliability Module). Because the Course is quite thorough, it will likely take as long as 40 hours to complete. Of course, if you are already somewhat familiar with simulation (and/or have a strong quantitative background), you may in fact be able to cover the material in considerably less than 40 hours. You can find the Course here: https://www.goldsim.com/Courses/BasicGoldSim/.

2. **Read "Getting Started with the Reliability Module"**. This is available both within the Help File and in the Reliability Module User's Guide (as Chapter 2). This provides an introduction to and a "quick tour" of the GoldSim Reliability Module. It presents the basic concepts of how risk and reliability can be simulated in GoldSim, provides an overview of the features and capabilities of the program, and summarizes the specialized elements associated with the module.

   Read more: Chapter 2: Getting Started with the Reliability Module (page 15).

3. **Request your free one hour web-based training session.** When you purchase GoldSim, you are entitled to a free one hour, live web-based training session in which one of our analysts provides an interactive training session via the Internet and telephone. You are strongly encouraged to take advantage of this free training, during which our analysts can provide an introduction to both the basic GoldSim framework and the Reliability Module.

4. **Open and explore the example files**. When you install GoldSim, a folder labeled "Reliability Examples" is installed with the program. (You can quickly access these files by selecting **File|Open Example…** from the main GoldSim menu). This directory contains example Reliability Module model files. These examples are introduced and discussed in Chapter 7. These example model files are an excellent way to begin to experiment with the Reliability Module.

   Read more: Chapter 7: Example Reliability Module Applications (page 145).

5. **Download Example Files from the Model Library**. The GoldSim website contains a Model Library with a number of models illustrating how GoldSim can be used for particular applications. These models tend to be more complex than the simple example files found in the Reliability Examples folder, but still relatively simple. Again, while exploring the files, use GoldSim's context-sensitive Help (i.e., the **Help** button in each dialog) to learn more about particular elements or features utilized in the model.

6. **Browse the User's Guide or Help System**. GoldSim has a large number of features, and you will not discover all of them by experimenting with simple example models. To fully utilize GoldSim's powerful features, browse through the User's Guides, using

the index and table of contents as your guide. Each section of the User's Guides is heavily cross-referenced, so it is easy to just jump around. Note that the Help system contains all of the contents of the User's Guides, with the exception of the technical appendices.

7. **Contact us with questions**. When you purchase GoldSim, you are entitled to one year of free support. This does not include assistance in building and debugging your models, but it does include answering questions on how to use GoldSim's features, so feel free to contact us! The best way to do so is through the GoldSim Help Center.

Conventions Used in this Manual

The following conventions are used in this manual:

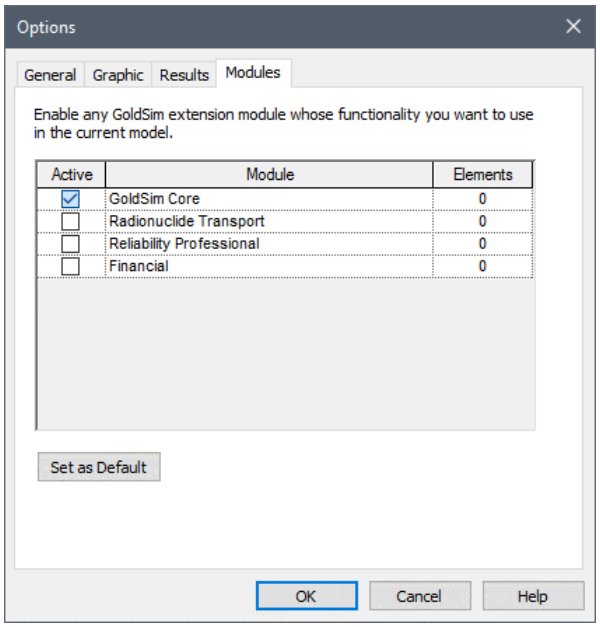
Convention	Description
Important Terms	New and important terms are presented in *bold italics*. These terms all appear in the Glossary of Terms at the end of the document.
File | Open…	Menus and menu selections are separated by a vertical bar. File | Open… means "Access the File menu and choose Open"
Failure Modes	Dialog buttons and tabs are identified in **Bold** font.
CTRL+C	Key combinations are shown using a "+" sign.. CTRL+C means press the Control and C keys simultaneously.
	**Warning**: This means watch out! Warnings typically alert you to potential pitfalls and problems that may occur if you perform (or fail to perform) a certain action.
	**Note**: Notes highlight important information about a particular concept, topic or procedure, such as limitations on how a particular feature can be used, or alternative ways of carrying out an action.

Activating the Reliability Module

In order to access the features and capabilities of the Reliability (RL) Module, either RL Learning Edition or RL Professional must be activated on your machine. Your copy of GoldSim will have one or the other (but not both). RL Learning Edition is automatically included with GoldSim Pro. RL Professional must be purchased separately.

Read more: What is the Difference Between Reliability Professional and Reliability Learning Edition? (page 6).

You can determine which version of the Reliability Module you have and whether ti is activated on your machine by selecting **Model| Options…** from the main menu, and selecting the **Modules** tab:



All extension modules that you are licensed to use appear in the dialog.

You can activate and deactivate modules that you are licensed to use by clicking the **Active** checkbox. By default, whenever you activate GoldSim, none of the available extension modules allowed by your license will be activated. To use them, you must activate them using this dialog.

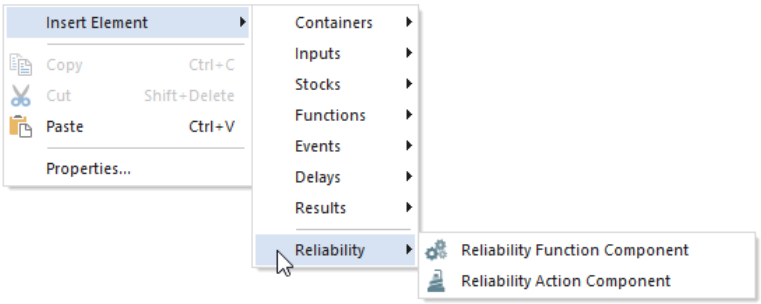
If you deactivate a module (such as the Reliability Module), any specialized elements associated with that module will be deleted (if any are present) and any menu options will be removed in the current file. If you make a module active, the various options associated with that module are made available again. If you press the **Set as Default** button, the selected modules will be activated for all new models that are created.

Note: If you try to open a file that contains more than 10 reliability elements, and you are licensed to use Reliability Professional but it is not currently active, GoldSim will automatically activate the module and open the file. If, however, you are not licensed to use Reliability Profession, GoldSim will not be able to open the file (and will display an error message).

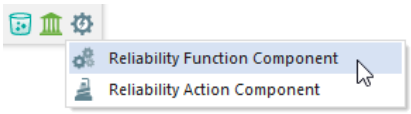
What Does the Reliability Module Add to the GoldSim User Interface?

The Reliability Module is a program extension to the GoldSim simulation framework. As such, it simply adds some additional features to the user interface. The user interface components which the Reliability Module adds can be summarized as follows:

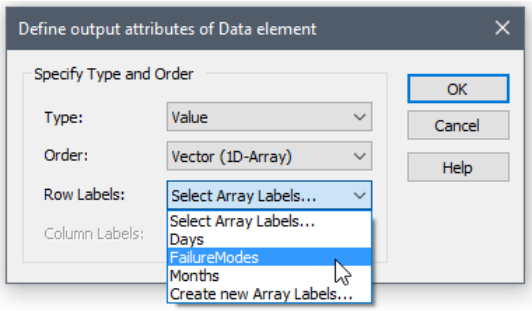
- An additional item, **Reliability**, is added to the context-sensitive (right-click) menu in the graphics pane (shown below), to the context-sensitive menu for Containers, and to the **Model| Insert** menu on the menu bar:



- Pressing the RL Element button in the Element toolbar provides access to a menu for inserting a RL Module element:

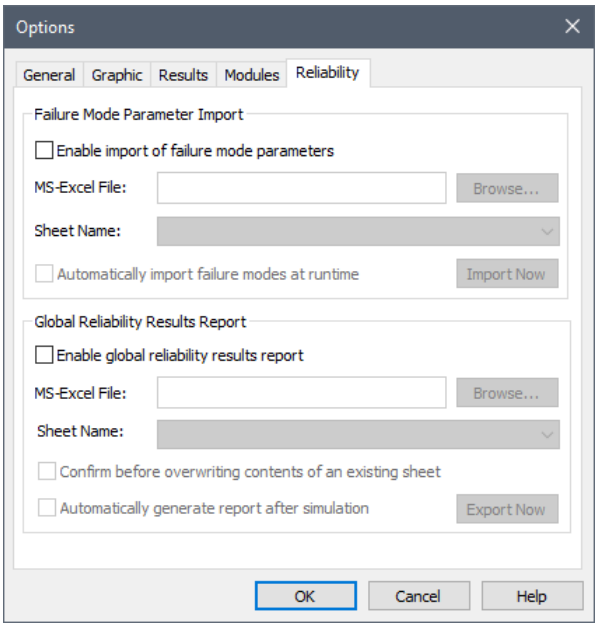


- The *FailureModes* array label set is added to the drop-down lists available when defining the output attributes for an element:



  This set of array labels is indexed from 1 to 10 (and cannot be edited), and is used to reference failure modes.

- A **Reliability** tab is added to the Options dialog (accessed via **Model | Options**) for defining options which are specific to the Reliability Module:



Technical Support, User Resources and Software Upgrades

The GoldSim Technology Group is dedicated to providing complete solutions for our customers. We pride ourselves in providing prompt and extensive support and resources to our users, and are committed to ensuring that each installation of our software is successful and adds value to the customer.

GoldSim Maintenance Program

When you purchase GoldSim software, you receive one year of Software Maintenance, entitling you to the following:

- Free software upgrades so that you always have the latest version of the GoldSim software.
- Basic Technical Support via email and phone. Basic support covers installation and licensing questions, as well as questions about GoldSim's features and capabilities.

After the first year, if you wish to continue to have access to new versions and technical support, Software Maintenance can be extended each year with payment of an annual fee.

Details regarding the GoldSim Maintenance Program can be found at www.goldsim.com/Web/Products/BuyGoldSim/Pricing/MaintenanceProgram/.

Getting Technical Support

Users with active Software Maintenance can submit questions directly to the GoldSim support team. Evaluation users are also welcome to contact us with questions on GoldSim functionality. The **GoldSim Help Center** (https://goldsim.zendesk.com) is the primary portal for technical support. You can submit your questions directly from the Help Center. If your register and log in through the Help Center, you will be able check the status and view a history of all of your support requests.

The Help Center also includes:

- The **GoldSim Forum**, where you can you can post questions to the GoldSim community, or just browse existing messages;

- Articles on licensing questions and modeling tips; and
- An archive of past webinars (which demonstrate GoldSim features and capabilities).

Free Basic Technical Support does not include consulting, model trouble-shooting or detailed assistance with applying GoldSim to a particular problem. Assistance of this nature is defined as Advanced Technical Support. Users may purchase Advanced Technical Support in pre-paid 10 hour blocks.

Details regarding Advanced Technical Support can be found at www.goldsim.com/Web/Resources/TechnicalSupport/.

Other GoldSim Resources

In addition to the GoldSim Help Center, additional resources are also available. These three resources can be accessed directly from the GoldSim website (www.goldsim.com):

- A free **Online Training Course** that will provide you with a thorough understanding of the key concepts on which GoldSim is based and all of the fundamentals required to build complex models of nearly any kind of system.
- The **GoldSim Model Library**, which contains a collection of example models to allow you to see how specific features of GoldSim can be used and/or how GoldSim can be used for specific applications.
- The **GoldSim Blog**, which provides an informal mechanism for GoldSim staff to share their knowledge, point out some of the more advanced (and perhaps overlooked) GoldSim features, share and discuss common mistakes we see in GoldSim applications, discuss interesting applications, and keep you abreast of our plans for further GoldSim developments.

You can stay up to date on the latest GoldSim news through these resources:

- The GoldSim LinkedIn Group, which is primarily used for announcements (e.g., new versions, interesting applications). You can join the Group here: www.linkedin.com/groups/1798413)
- Periodic email newsletters are sent two to three times per year. To be added to the newsletter list, contact us via the GoldSim Help Center (https://goldsim.zendesk.com).

Note: When you purchase GoldSim, you are entitled to a free one hour, live web-based training session in which one of our analysts provides an interactive training session via the Internet and telephone. You are strongly encouraged to take advantage of this free training.

References

The references cited in this chapter are listed below:

Ebeling, C.E, 1997, An Introduction to Reliability and Maintainability Engineering, McGraw-Hill, Boston.

Stamatelatos, M. et al., 2011, Probabilistic Risk Assessment Procedures Guide for NASA Managers and Practitioners, Second Edition, NASA/SP-2011-3421, NASA Headquarters, Washington, D.C.

Vesely, W. et al., 2002, Fault Tree Handbook with Aerospace Applications, NASA Office of Safety and Mission Assurance.

Chapter 2: Getting Started with the Reliability Module

> A 10 cent fuse will protect itself by destroying the $2000 radio to which it is attached.
>
> Robert Livingston, *Flying the Aeronca*

Chapter Overview

This chapter provides a brief overview of some key concepts of the GoldSim simulation framework, provides an overview of the features and capabilities of the Reliability Module, and walks you through a simple example in order to help you get started.

If you read nothing else before starting to use the Reliability Module, it is strongly recommended that you read this chapter, as it will tell you what the program is capable of doing, provide an overview of how to build a model, and direct you to those portions of the manual where you can obtain further information.

Note: While this chapter provides an introduction to the features of the Reliability Module, the technical details are spelled out in Chapters 3 through 6.

In this Chapter

This chapter discusses the following:

- Basic GoldSim Concepts Necessary to Use the Reliability Module
- Overview of the GoldSim Approach to Risk and Reliability Modeling
- A Simple Reliability Module Example

Basic GoldSim Concepts Necessary to Use the Reliability Module

The Reliability Module is a program extension to the GoldSim simulation framework. In order to take full advantage of all of the powerful features of the Reliability Module, you will eventually need to become familiar with many of the features and capabilities of the underlying GoldSim simulation framework.

At a minimum, in order to get started with the Reliability Module, you must have an understanding of some fundamental concepts regarding GoldSim. These concepts are provided in the GoldSim Tutorial, which can be accessed from the opening splash screen of GoldSim, or by pressing **Help | Tutorial…** from the main menu. The Tutorial takes approximately one to two hours to complete (and can be completed in phases). If you are new to GoldSim and have not yet completed the Tutorial, you should do so now before proceeding.

Summary of Basic Concepts

The basic GoldSim concepts that should be understood before you get started with the Reliability Module are summarized below. These basic concepts are all discussed in the GoldSim Tutorial, and fall into the following categories:

- Basic Simulation Concepts
- Basic GoldSim Concepts
- Introduction to the GoldSim User Interface

The concepts are summarized below. If you are not comfortable with these concepts, you should return to the Tutorial and/or consult the portions of the **GoldSim User's Guide** or Help file that discuss these concepts (and are cross referenced in each of the three summary sections listed below).

Throughout the remainder of this document, whenever necessary, these features and capabilities are mentioned and discussed briefly. Cross-references are provided to the more detailed descriptions available in the **GoldSim User's Guide**.

Basic Simulation Concepts

A summary of basic simulation concepts discussed in the GoldSim Tutorial is provided below:

- Simulation is the process of creating a model of an existing or proposed system in order to identify and understand the factors that control the system, or to predict the future behavior of the system.
- In a static simulation, the system model does not change with time.
- In a dynamic simulation, the system model changes and evolves with time.
- Deterministic simulation often represents "the best guess" or "worst case" input values.
- Probabilistic simulation represents uncertainty and randomness by specifying some inputs as probability distributions or random events.
- Simulation is a powerful and important tool because it provides a way in which alternative designs, plans and/or policies can be evaluated without having to experiment on a real system.

Basic GoldSim Concepts

A summary of basic GoldSim concepts discussed in the GoldSim Tutorial is provided below:

- GoldSim represents parameters, processes, or events in a system using objects called elements.
- Each element has a symbol or graphical image to represent it, and has a dialog where you specify its properties.
- An element accepts input data and produces output data.
- There are six primary categories of elements: Inputs, Functions, Events, Stocks, Delays, and Results.
- GoldSim represents the links (dependencies) between elements using arrows called influences.
- A special type of element, the Container, can be used to hierarchically organize other elements.
- GoldSim ensures dimensional consistency and carries out all unit conversions for you.

These topics are discussed in detail in Chapter 3 of the **GoldSim User's Guide**.

Introduction to the GoldSim User Interface

A summary of basic GoldSim user interface concepts discussed in the GoldSim Tutorial is provided below:

- GoldSim has two sections to its window: the graphics pane and the browser.
- By default, the browser is displayed on the left side of the GoldSim window. You can open and close the browser using the browser button on the toolbar:

- GoldSim commands can be accessed using the menu, toolbars, or context menus.
- Elements are added to the graphics pane by right-clicking on an empty section of the graphics pane, and selecting the appropriate element from the context menu.
- Element inputs, outputs and settings are specified in the element's Properties dialog, which can be displayed by double-clicking on the element's icon.
- The Simulation Settings dialog can be displayed by selecting **Run|Simulation Settings** from the main menu, by pressing **F2**, or by pressing the Simulation Settings button in the toolbar:

  The Simulation Settings dialog allows you to adjust the length of the simulation, the timestep length, and the number of Monte Carlo realizations.
- A model is run by by selecting **Run|Run Model** from the main menu, by pressing **F5**, or by pressing the Run button in the toolbar:

- After a model is run, it is in Result Mode. You cannot change the structure of the model or any of the input values while in Result Mode.

- Results can be viewed by right-clicking on an element, or double-clicking on a Result element.
- You can return to Edit Mode from Result Mode by selecting **Run|Return to Edit Mode**, pressing **F4**, pressing the Edit Mode button in the toolbar:

These topics are discussed in detail in Chapter 3 of the **GoldSim User's Guide**.

Understanding Discrete Events and Triggering

Use of the Reliability Module requires a good understanding of one set of advanced features in GoldSim: discrete event modeling. Discrete event modeling is discussed briefly in the Tutorial, but due to its importance within the context of the Reliability Module, it is discussed further here.

This basic description of discrete event modeling and triggering presented here should be sufficient to allow you to understand the features and capabilities of the reliability elements discussed in subsequent sections.

However, to fully utilize all the features of GoldSim and the Reliability Module, you will eventually want to learn more about the details of discrete event modeling, and can do so by consulting the portions of the **GoldSim User's Guide** or Help file that discuss these concepts.

Discrete Event simulation is discussed in detail in Chapter 5 of the **GoldSim User's Guide**.

What is an Event?

In GoldSim, a discrete event is something that occurs instantaneously (as opposed to continuously or gradually) in time. It represents a "spike", a discontinuity, a command, or a discrete change of state for the system.

For example, through continuous compounding of interest, the money in a bank account continuously increases, but the account can also increase and decrease instantaneously due to discrete events (i.e., deposits and withdrawals).

Such events are particularly important for the Reliability Module because the systems being modeling are controlled primarily by discrete occurrences (such as failures and repairs).

Of course, "instantaneous" and "gradual" are relative terms. That is, whether something is treated as instantaneous or gradual is a function of the time scale of interest, and hence you must differentiate between the two based on the context of your model. Typically, the distinction will be obvious. For example, if the time scale of interest is 10 years, something happening over the span of a day can be considered to be "instantaneous". If the time scale of interest is several days, however, something happening over the span of a day would in most cases need to be treated in a continuous manner.

GoldSim handles "instantaneous" changes to a model by providing a mechanism for a model to generate and respond to discrete events. This is accomplished by providing the ability to instantaneously trigger an element to take a particular action (e.g., instantaneously change its value) in response to an event. Hence, in GoldSim, an event is specifically defined as *an instantaneous occurrence that subsequently triggers a particular action*.

In GoldSim, an event can be generated in one of four ways:

- The event occurs when a specified condition (e.g., X > Y) becomes true or false;
- The event occurs when a specified output in the model changes;

- The event occurs at a specified calendar or elapsed time; or
- The event occurs based on a specified rate of occurrence, which can be treated as regular or random ("occur exactly once a week" or "occur, *on average*, once a week").

In addition, some elements can respond to an event generated via one of the mechanisms above, and generate a new event.

In some cases, an event will occur (e.g., X becoming greater than Y) which triggers a particular action in a single element (exercise an option). In such a case, the event is internal to that element, and it does not directly impact other elements. In other cases, however, an event may impact multiple elements, or one element may respond to an event by triggering other elements to take a particular action. In these cases, it is necessary for discrete signals to propagate between elements.

In order to propagate events (and their consequences) between elements in a model, it is necessary to send information between elements intermittently as a "spike" or discrete "packet" of information. To facilitate this, GoldSim allows certain elements to emit and receive (i.e., produce as outputs and/or accept as inputs) a ***discrete signal***. Discrete signals are a special category of outputs that emit information discretely, rather than continuously.

Within GoldSim, there are actually two types of discrete signals that can be passed from one element to another: discrete event signals and discrete change signals.

A ***discrete event signal*** is a discrete signal indicating that something (e.g., a purchase or a sale) has occurred. It does not describe the consequence of that occurrence; it simply emits a signal between elements indicating that an event has occurred.

A ***discrete change signal***, on the other hand, emits information regarding the response to an event. In particular, a discrete change signal contains two pieces of information: a value (e.g., 10 dollars) and an instruction (e.g., Add). A discrete change signal can only be generated in response to an event. It can only be received by a few select elements (e.g., a Reservoir or a Pool) which understand how to process it.

Within the Reliability Module, discrete event modeling is a fundamental feature of the elements used to model reliability and risk, and is utilized in the following ways:

- Reliability elements (which are used to model system components such as pumps, engines, relays, and switches) can be triggered to turn on and off. They can also be triggered to be replaced or repaired.
- One class of Reliability elements (Action elements) are used to model components which must respond to a control command or condition (e.g., switches, relays). Events provide the command or condition that tells the element to carry out its action.
- Reliability elements output discrete event signals to mark a number of occurrences, such as whether or not a requested action was successful and when a component starts or stops operating. These events can then be used to subsequently trigger other elements.

Basic GoldSim Elements that Support Modeling of Events

In addition to elements within the Reliability Module, GoldSim provides a wide variety of other elements that can be triggered by events or can output discrete event signals. These elements are important, as they can be used in conjunction with the reliability elements to represent complex risk and reliability models.

For example, the outputs of these elements can trigger reliability elements to turn on or off. In other cases, discrete event signals from reliability elements may trigger them to perform a particular action (set a Status element to True or False).

Some of the event elements that are likely to be most useful to risk and reliability modelers are summarized in the following table:

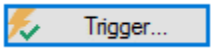
Element	Default Symbol	Function
Timed Event		Generates events based on a specified rate of occurrence, regularly or according to a specified distribution (i.e., randomly).
Triggered Event		Generates events based on one or more specified conditions.
Decision		Generates one of up to three alternative events based on specified conditions.
Random Choice		Generates a user defined event based on specified probabilities.
Milestone		Records the time at which a particular event or specified condition(s) occurs.
Status		Outputs a condition (True/False) in response to particular events or specified conditions.
Event Delay		Delays an event for a specified time. Can also be used to simulate queues.

Note that elements such as the Triggered and Timed Events are often used to create initiating events (e.g., events external to the system being modeled, such as an earthquake or a random interruption) that can lead to the failure of a system component.

Defining Triggers

All discrete event modeling involves *triggering* of one form or another. When you trigger an element, you are telling it that a discrete event has occurred that you want the element to respond to.

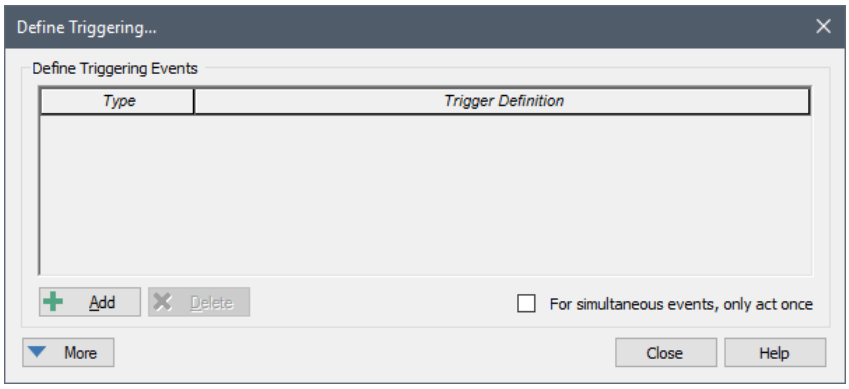
Various elements respond to a trigger in different ways. They are, however, all triggered in the same way. That is, these elements all share a common Triggering dialog, which controls how they are triggered. All Triggering dialogs are accessed via a **Trigger…** button that will look similar to this:



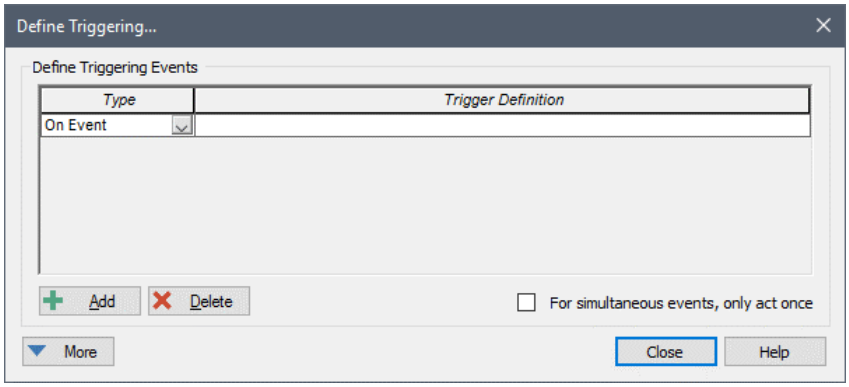
Note: Depending on the element, the label for the trigger button will not always be "Trigger…".

Note: Holding your cursor over the trigger button displays a tool-tip summarizing the current trigger settings.

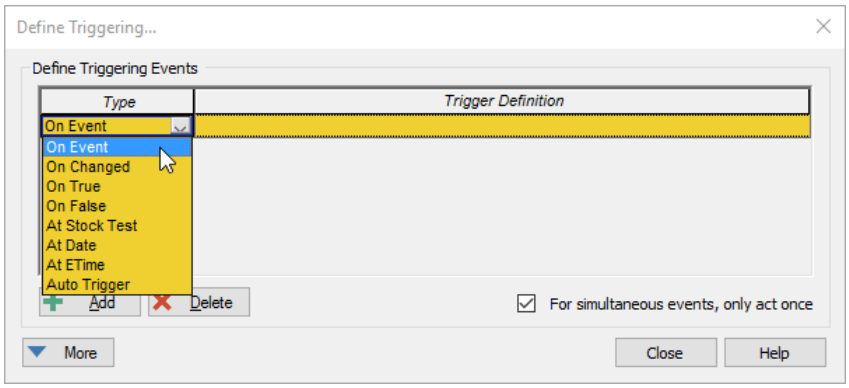
The Triggering dialog for every element looks like this:



To define a trigger for the element, you simply press the **Add** button to add a row to the list of triggering events:



By default, the **Type** of event added will be "On Event". If you click on the small button in the **Type** column, a drop-list will be presented allowing you to edit the event type:



There are eight types of events that can be added:

On Event: The Trigger Definition must be a discrete event signal from another element. The element is triggered whenever the signal is received.

On Changed: The Trigger Definition can be any continuous output (it cannot be an expression or a discrete signal). The element is triggered whenever the value of Trigger Definition changes.

On True: The Trigger Definition can be any condition output or conditional expression. The element is triggered whenever the Trigger Definition *becomes* True.

On False: The Trigger Definition can be any condition output or conditional expression. The element is triggered whenever the Trigger Definition *becomes* False.

At Stock Test: The Trigger Definition must be a conditional expression of of the form "A>B", "A>= B", "A<B", "A<=B", or "A=B" where A is the primary output of a Reservoir, a Pool or a Fund (from the Financial Module). The element is triggered whenever the Trigger Definition *becomes* True. As discussed below, this triggering event interrupts the clock and adds an unscheduled update.

At Date: The Trigger Definition must be a date or date and time, enclosed in quotations. (Alternatively, the date can also be expressed as the amount of time since 30 December 1899). The element is triggered whenever the simulated date reaches the specified date. As discussed below, this triggering event interrupts the clock and adds an unscheduled update.

At ETime: The Trigger Definition must be an elapsed time. The element is triggered whenever the simulated elapsed time reaches the specified elapsed time. As discussed below, this triggering event interrupts the clock and adds an unscheduled update.

Auto Trigger (only available for some elements): An Auto Trigger requires no user-defined Trigger Definition and its behavior is defined by its context (i.e., the type of element). Auto Triggers react to the activation or deactivation of their parent Container.

Note: *At Stock Test*, *At Date* and *At ETime* triggers interrupt the clock and insert an unscheduled update when they occur (whereas *On True*, *On False* and *On Changed* triggers do not create an unscheduled update). To understand the implications of this, consider an example in which your scheduled updates were every 10 days. There are two different ways you could try to trigger an event when the value of Reservoir A became greater than the value B. You could create an *At Stock Test* trigger of A > B, or you could create an *On True* trigger of A > B. If we assume that A > B actually became true at 15 days, these two triggers would behave very differently. The *At Stock Test* trigger would catch this point exactly, and insert an unscheduled update at 15 days. In the absence of any other unscheduled updates, however, the *On True* trigger would not be evaluated and implemented until 20 days. Similarly, if you wished to trigger an event at a specific elapsed time (e.g., 17 days), you could try to do so in two different ways. You could trigger the event using an *At ETime* trigger of 17 days, or you could create an *On True* trigger with ETime >=17days. Again, these two triggers would behave very differently. The *At ETime* trigger would catch this point exactly, and insert an unscheduled update at 17 days. In the absence of any other unscheduled updates, however, the *On True* trigger would not be evaluated and implemented until 20 days.

The **More** button provides access to advanced triggering options.

Overview of the GoldSim Approach to Risk and Reliability Modeling

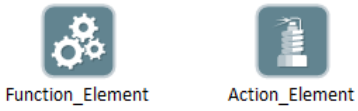
The Reliability Module uses an approach that is substantially different from most current approaches to investigating system reliability and risk. This section is intended to provide you with a broad overview of the GoldSim reliability modeling and risk analysis approach.

The goal of most reliability and engineering risk analysis models is to establish the lifespan, reliability, degree of availability, or probability of failure of a system. But how do we model a system's performance in GoldSim? Below we introduce the GoldSim approach to reliability and risk modeling by considering how one might model the reliability of a personal computer.

The Reliability Elements

The first step to modeling reliability in GoldSim, as it is in any other reliability and risk analysis modeling methodology, is to develop a model of the system of interest with all of its components. In GoldSim, the building blocks used to represent the components of the system are the reliability elements themselves.

GoldSim provides two types of reliability elements: the Function element and the Action element:



Function elements are used to model components which operate continuously once turned on. Typical examples of components modeled by Function elements include pumps and engines.

Action elements are typically used to represent components which must respond to a control command or condition. Typical examples of components modeled by Action elements include switches and relays.

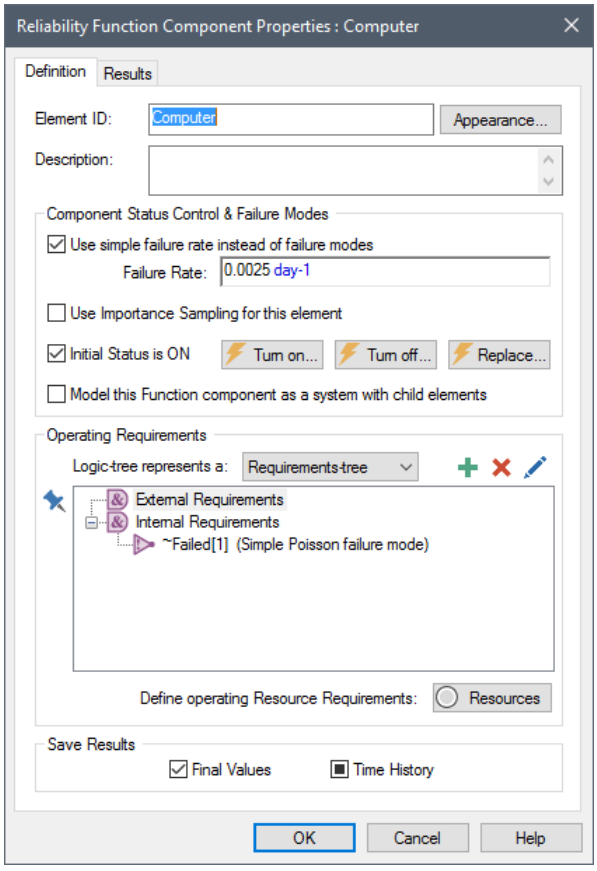
Both element types can fail, as well as be repaired and maintained. In addition, the Action element has the ability to determine whether or not a requested action has been completed in a satisfactory manner.

Read more: Chapter 3: The Reliability Elements (page 49).

Top-Down Modeling Using the Reliability Module

GoldSim encourages a top-down approach to modeling. In a top-down approach, the system is initially represented in a simple manner, and then evolves (and becomes more detailed and realistic) as more information about the system is obtained. A top-down approach to modeling is inherently an iterative approach. To illustrate this, let's consider an example of modeling the reliability of a computer.

Following a top-down approach, our first attempt at modeling the reliability of the computer might be very simple. In particular, we might begin by using a single Function element to represent the computer, and assign an exponential (constant) failure rate to model the computer's reliability. The Function element's dialog for this simple representation is shown below:



In this particular case, we have specified that the reliability of the computer can be modeled by assuming that it has an exponential failure rate, with a mean of 1/400 days (the **Failure Rate** input field is set to 0.0025 day-1, or 1/400 days).

A quick look at the dialog reveals a number of options and fields which can be used to incorporate additional detail and complexity into the model as the system becomes better quantified. Once a preliminary model has been developed and run, users will typically develop an improved, more realistic model using some of the more advanced features of the Reliability Module.

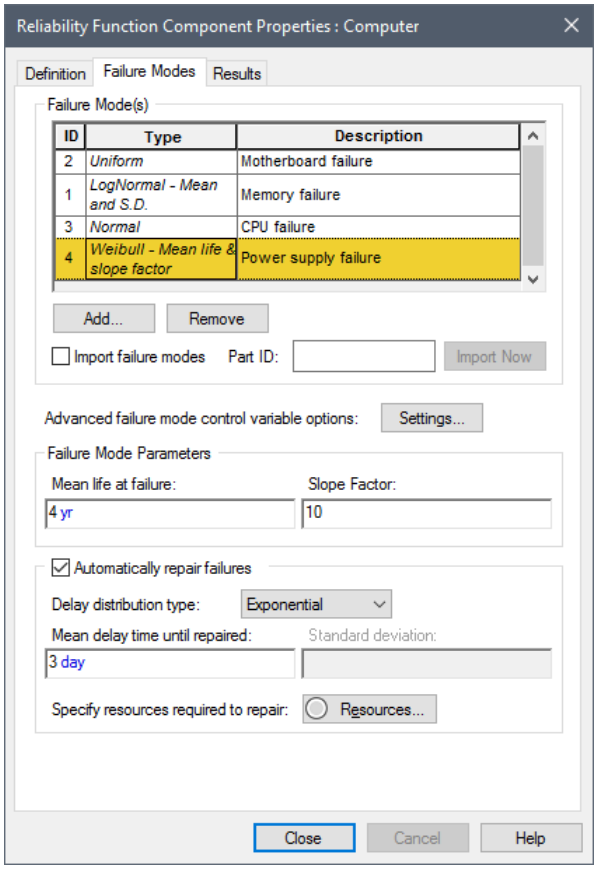
We will illustrate some of these advanced features below as we add some detail to our reliability model for the computer.

Read more: Overview of the Function Element (page 50).

Adding Failure Modes to a Reliability Element

There are a number of ways that a computer could fail. Hence, lumping these various failure modes into a single exponential rate is surely a simplistic approach. Furthermore, our simple model does not represent repairs. If our objective is to predict the availability of the computer, we need to simulate repairs too.

To facilitate this, GoldSim allows us to move beyond a simple exponential failure rate, and specify multiple distinct and independent failure modes. In the dialog below, we have specified five separate failure modes for the computer:



Each failure mode represents a particular type of failure (e.g., failure of the power supply), and each type of failure has its own distribution, which does not have to be exponential. For example, the Power Supply failure mode uses a Weibull distribution (a common failure distribution for reliability models).

Each failure mode can also be specified to be automatically repaired (with the time until it is repaired being different depending on the failure mode). In the example above, if the computer fails due to the Power Supply failure mode, it is assumed to be repaired in approximately 3 days (repair time is modeled using a distribution, and is therefore variable).

It is important to note that the ***failure mode control variable*** (i.e., the variable that is referenced by the failure mode to determine when failure occurs) for failure modes does not have to be absolute or operational time. In the case of a car you might build a model that uses the car's mileage to determine when

failure occurs, while in the case of an airplane, you might base failure modes on the number of takeoffs and landings. In these cases, other elements in your GoldSim model would be used to model the cumulative mileage or cumulative number of takeoffs and landings. In fact, this is one of the most powerful features of the Reliability Module; because it is part of a general-purpose simulation framework it can incorporate many factors into the analysis that other approaches cannot.

Similarly, failure mode parameters do not have to be constant during the course of the simulation. You might change the failure mode parameters to reflect the current state of the system, (e.g., you could specify that a failure of a car's rear braking system increases the wear on its front braking system), or to reflect the effects of the operating environment (e.g., a computer's failure rates could be influenced by the temperature and humidity of the room).

Note: By default, failures are assumed to be fatal to the component. That is, if a particular failure mode occurs, the component itself fails and is no longer operative. However, if desired, you can specify that a failure mode is non-fatal in order to model more complex failure mode behavior.

Read more: Defining Failure Modes (page 92); Modeling Coupled and Non-Fatal Failure Modes (page 105).

Modeling Hierarchical Systems of Components

Although modeling a computer using multiple failure modes is an improvement over using a single exponential failure mode, perhaps we determine that modeling something as complex as a computer requires us to model the individual components of the computer separately.

Because GoldSim and the Reliability Module are hierarchical by nature, we can quickly transform our simple single element model into a system with a number of subcomponents. Again, the intention is to allow our model to evolve (and hopefully improve) as the system becomes better quantified.

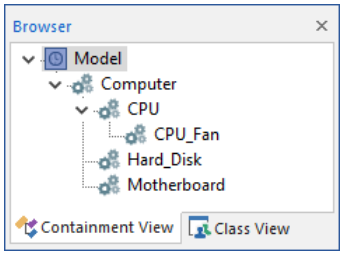
In this case, we can convert the model of the computer into a system with child (sub) elements. (On the main page of the Function dialog, there is an option to **Model this Function component as a system with child elements**). When we do this, there is a slight change in the appearance of the Computer element:

A small red triangle now appears in the upper left-hand corner of the element, and clicking on this takes you into a new layer of the model *within the computer*. If you are familiar with GoldSim, you will recognize this triangle as the same symbol that normally appears on GoldSim Container elements. In fact, when we convert a reliability element into a system, the element retains all of its properties, but now also functions as a Container. (In fact, it actually becomes a ***localized Container***).

Note: Localized Containers are discussed in detail in Chapter 10 of the **GoldSim User's Guide**.

In our example, we might choose to place a CPU, a Motherboard and a Hard Disk inside the Computer element, and model each of these components in as much detail as we wish:

Note, however, we are not limited to one level of hierarchy in GoldSim. In fact, we can create an unlimited number of subcontainers (subcomponents). In the figure below, a browser view of the model is provided, showing the hierarchy of components. In this case, a Function element representing a CPU Fan has been placed inside the CPU Function element, adding another level to the model hierarchy:



Of course, each of these subcomponents can have its own failure modes.

What are the benefits of using hierarchical modeling in order to model a reliability system? The most obvious is that it makes the model much more intuitive and easy to browse. If we use hierarchical modeling, we have just one element in the top level of the model for the computer, where we would otherwise have five. While this may be a subtle distinction with just one computer, it is a feature which becomes very valuable when one needs to model a bank of 50 computer servers.

In addition, due to the logical relationships that exist between components that are actually hierarchical in the real world (e.g., a computer contains a CPU which contains a CPU fan), it may not be possible to represent some kinds of systems in a realistic or straightforward manner without defining such a hierarchy. This is discussed further below.

Read more: Modeling a Reliability Element as a System with Child Elements (page 58).

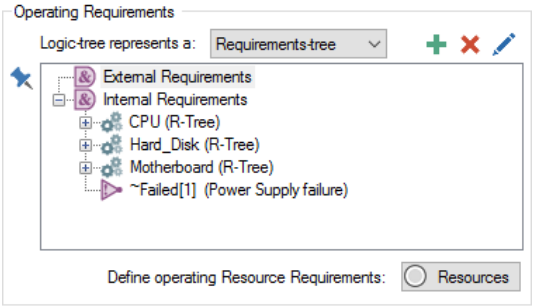
Representing Logical Relationships Between Components

In nearly all multi-component systems, logical relationships will exist between the components (e.g., "in order for component A to operate, component B or component C must be operating"). These relationships can be represented within the reliability elements by defining either a ***requirements-tree*** or a ***fault-tree***. In a requirements-tree, the top level node must evaluate to true for the component to operate, and for a fault-tree, the top level node must evaluate to false for the component to operate. These trees are used to model the logical relationships between components, and to propagate the effects of changes in element status to affected components.

In the case of the Computer element, it requires an operational CPU, motherboard and hard disk in order to function properly. In the case of the CPU, it needs its fan to be operational in order to function properly. Of course, each of these subcomponents can have its own set of failure modes. Let's assume for the purpose of this example that all of the failure modes for the Computer (except one) are assigned to the subcomponents. One failure mode

(Power Supply Failure) is assigned to the parent element (the Computer), since we have decided not to explicitly represent the power supply as another element.

GoldSim can easily represent these relationships, but it provides a slightly different approach to fault trees than many reliability engineers will be used to. In particular, there is no need to develop a "global" requirements- or fault- tree for the entire system. The user is only required to define the immediate relationships between components. In this case, the computer's requirements tree would include four nodes, indicating that in order for the Computer to operate, the CPU, Hard Disk and Motherboard must all be operating, and the Computer itself must not have failed due to the Power Supply failure mode:



Note that these are listed as "Internal Requirements" because the components and the failure mode are internal to the Computer. Failure modes for the Computer itself are added automatically to this list by GoldSim. Any sub-components need to be added manually (as appropriate).

"External Requirements" can be used to specify requirements external to the Computer (e.g., that electricity is available).

Read more: Failure Modes and Internal Requirements (page 95); Defining Operating Requirements for Reliability Elements Using Logic Trees (page 64).

Using GoldSim's Probabilistic Simulation Engine

Once a model of the Computer has been constructed, we can simulate the system to predict how it will perform through time. By definition, however, the performance of such a system is stochastic (i.e., inherently variable). That is, we can't say exactly when a component will fail; we can only describe the failure (and repair) process statistically. For example, if we had 100 identical computers, their failure (and repair) histories would not be identical. They would display a distribution of behaviors.

In addition to this inherent variability, we might also be uncertain about some of the input parameters controlling the model. For example, if we had not carried out actual tests on the components, the parameters describing their failure modes would be uncertain, and we could enter these as probability distributions in order to capture this uncertainty.

Variability and uncertainty are represented in GoldSim using Monte Carlo simulation. Monte Carlo simulation consists of a calculating a large number of "realizations" (potential futures). In our computer example, this would be equivalent to simulating the behavior of a large number of computers through time.

Read more: Using Monte Carlo Simulation in Your Reliability Model (page 84).

Viewing and Analyzing Results

GoldSim provides a number of reliability analysis tools that become available at the end of a simulation.

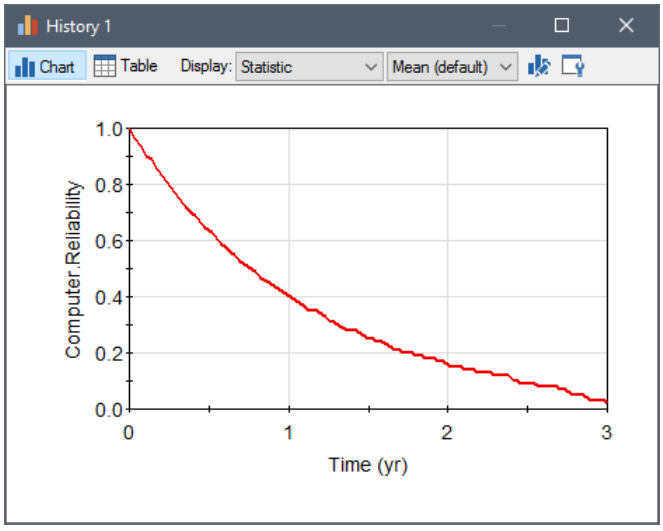
The most fundamental result is the summary of the mean reliability and availability statistics over the duration of the simulation:

Summary

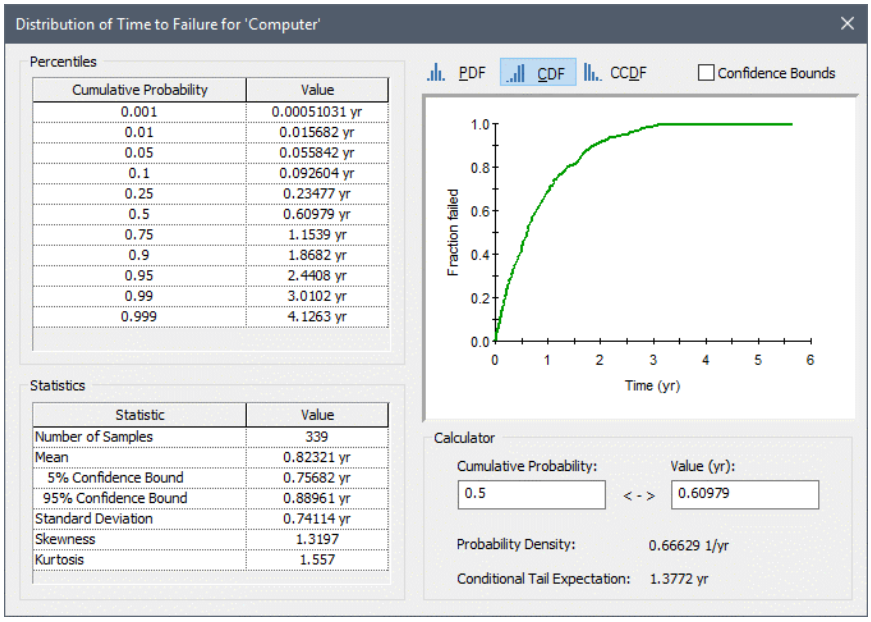
Results for 100 realizations with mean duration of 3 yr.

Measure	Confidence Bounds		
	5%	Mean	95%
Operational Availability:	0.930	0.943	0.956
Inherent Availability:	0.930	0.943	0.956
Reliability:	0.004	0.020	0.047

A wealth of additional information is also available. For example, a time history of the mean reliability can be displayed:

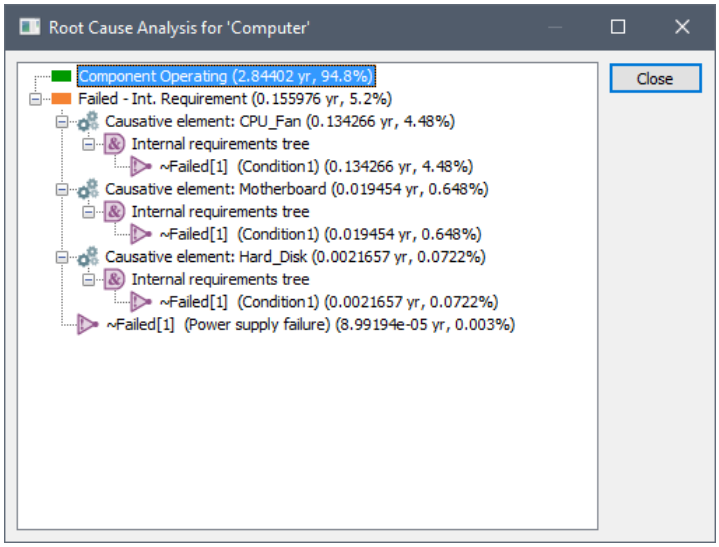


Also, statistical analyses of failure and repair time distributions can be displayed:

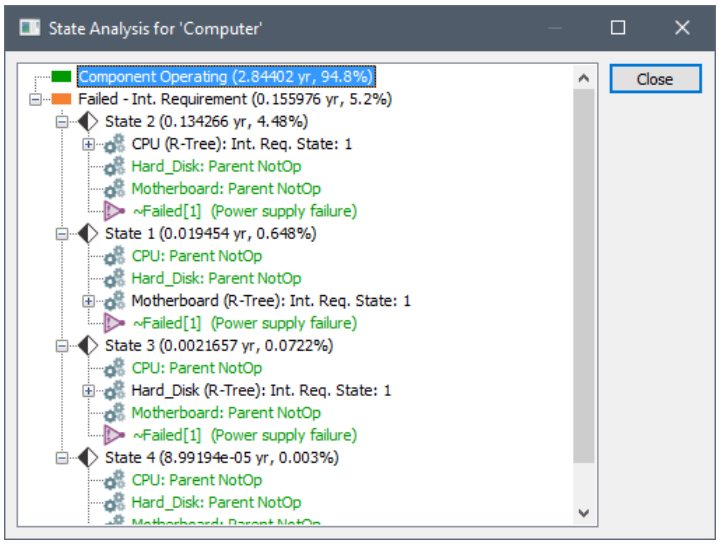


One of the most powerful types of result processing available within the Reliability Module is ***causal analysis.*** Because GoldSim records all of the unique conditions which result in failure of each individual element during a

simulation, it can provide analyses that identify failure scenarios and failure root causes for each reliability element:



A related analysis shows the different failure scenarios for a system, where each scenario corresponds to a specific state of the system's components. Each state is presented as a logic-tree, with each component that is not operating shown highlighted (in red):



Read more: Chapter 6: Displaying Reliability Results (page 129).

Documenting Your Reliability Model

GoldSim provides tools that allow you to internally document your model such that the documentation becomes part of the model itself, and hence is immediately available to anyone viewing the model.

GoldSim allows you to add text, images and other graphic objects directly to your model. In addition, you can add hyperlinks to other documents (e.g., a web site or a report). Clicking on the hyperlink then opens that document.

As discussed previously, GoldSim was specifically designed to allow you to organize model elements into a hierarchy (using *Containers*). This facilitates the creation of "top-down" models, in which the level of detail increases as you "push down" into the containment hierarchy. Such a capability is essential if you

wish to effectively describe and explain your model at different levels of detail to different audiences. For example, your manager may only want to see the "big picture", while a technical colleague may want to see the low-level details of a particular model.

The ability to create hierarchical, top-down models, in which at any level, details can be "hidden" (inside containers), coupled with GoldSim's powerful documentation features, allows you to design models which can be effectively explained to any audience at the appropriate level of detail.

Note: Techniques and tools for documenting GoldSim models are discussed in detail in Chapter 9 of the **GoldSim User's Guide**.

A Simple Reliability Module Example

Having provided a conceptual overview of the Reliability Module, this section provides a more hands-on introduction to the same concepts by walking through the steps necessary to build and edit a simple reliability model. Although it is not necessary to do so, it will be helpful if you actually follow along and build and edit the simple model as it is discussed below.

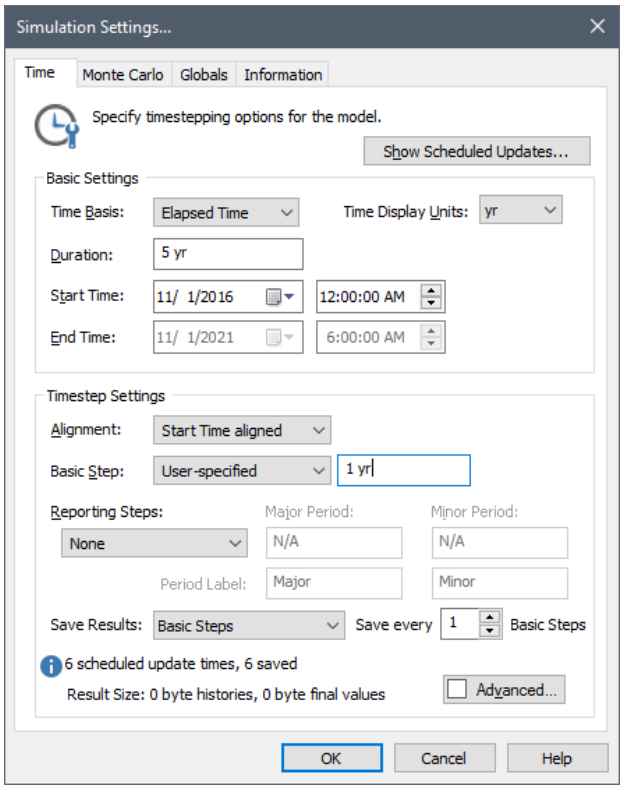
In order to do so, however, you must be able to carry out the basic GoldSim actions discussed in the GoldSim Tutorial (e.g., adding elements, modifying simulation settings, running a model). Therefore, if you wish to build and edit the model described below, and you have not already taken the Tutorial, you should do so now. The GoldSim Tutorial can be accessed by clicking the **Tutorial** link on the GoldSim splash screen, or by selecting **Help**|**Tutorial…** from the GoldSim main menu.

Step 1: Creating a Dynamic Reliability Model

One of the key distinctions between traditional reliability approaches and the GoldSim reliability approach is the way that time is treated. In many traditional reliability approaches, such as closed-form reliability equations, time is nothing more than a variable in the equation representing the service life of the component. GoldSim is quite different in that it is a dynamic simulator, meaning that events can occur, conditions can change and components can interact as time progresses over the course of the simulation.

Let's explore this concept by constructing a simple GoldSim model. If you wish to follow along, you can start to do so now by opening GoldSim (or if GoldSim is already open, by creating a new model by pressing the new model button or **Ctrl+N**).

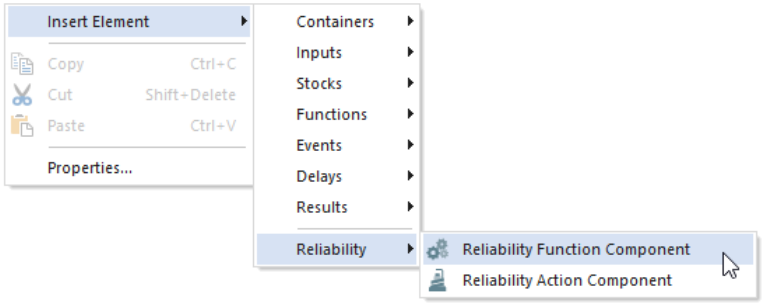
Let's start by specifying the simulation settings (by pressing **F2**). When specifying the simulation settings, we will specify that the simulation will be an Elapsed Time simulation with a duration of five years (5 yr), using a 1 yr **Basic Step**. Define the **Time Display Units** in years (yr):



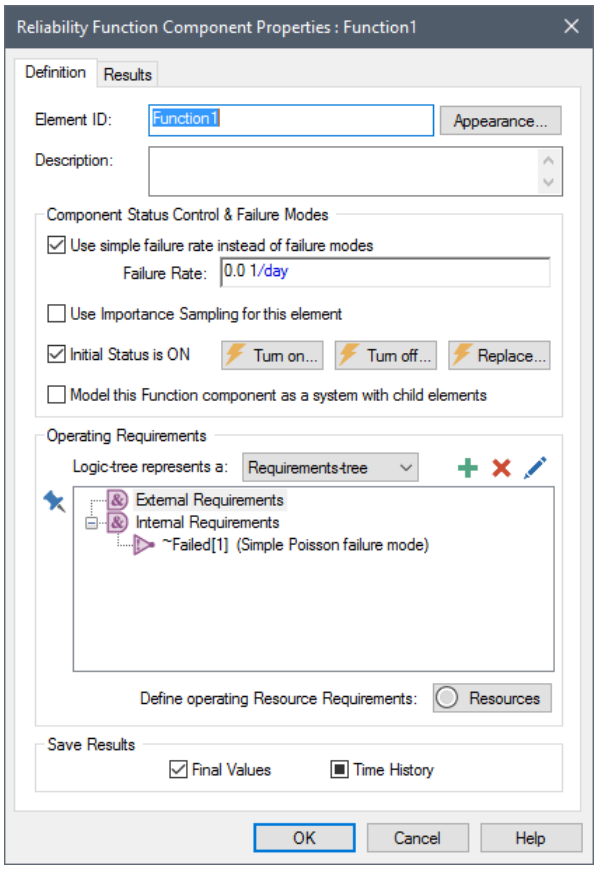
At this point, we do not need to edit the **Monte Carlo** tab, as we will start by running a single realization (the default).

Step 2: Adding a Reliability Function Element

Let's start by insert a new Function reliability element into the new model. We can do this by right-clicking in the graphics pane, and selecting **Insert Element|Reliability|Function Component** from the context menu:



This will insert the element and open the Function element's Properties dialog:

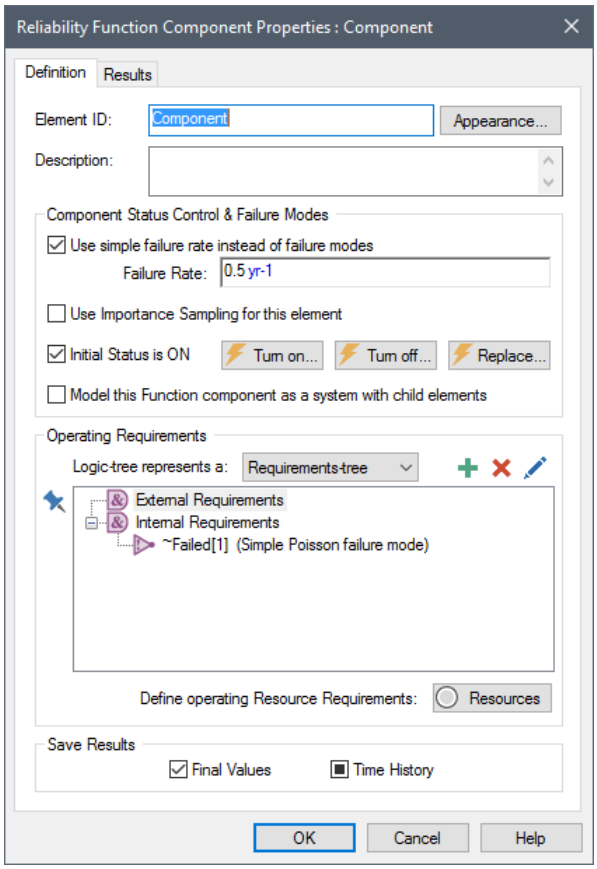


If this is your first close look at a reliability element, you will notice that it has some common features with the other basic GoldSim elements that you've seen in the GoldSim Tutorial. The **Element ID** and **Description** fields are identical (and the same naming rules apply to reliability elements as to normal GoldSim elements). The **Appearance** button and Save Results checkboxes are also common to all GoldSim elements.

Below the **Description** field, there is a section called "Component Status Control & Failure Modes". This sections allows you to define the failure behavior for the component (whether you use a simple, unrepaired exponential failure mode, or more advanced failure modeling), and buttons that allow you to set up triggering logic to turn the component on and off, or replace the component with a new one. It also contains an option to control whether or not Importance Sampling is used (a technique that can be used to facilitate representation of failure modes that occur very infrequently).

At the bottom of the dialog, there is an "Operating Requirements" section that allows you to specify logical relationships and conditions for the successful operation of the element. Note that it automatically includes an Internal Requirement that the default failure mode (Simple Poisson failure mode) has not occurred.

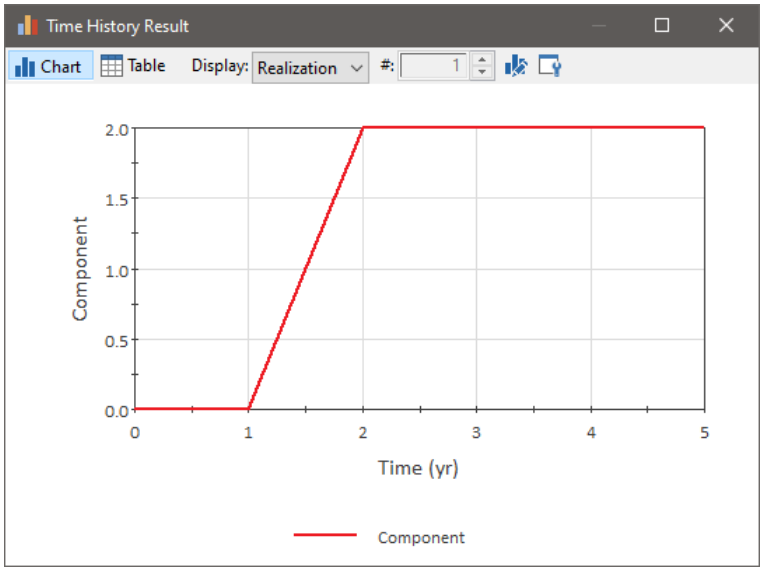
Now that we've taken a quick look at the Function element, let's change the **Element ID** (simply call it "Component"), and specify a simple failure rate of 0.5 yr-1 (equivalent to a mean time to failure of 2 years):



We now have a component with a mean failure time of 2 years, and a service life (the duration of the simulation) equal to five years. Press **OK** to close the dialog.

Step 3: Running the Model and Viewing a Simple Result

We can now run the model by pressing **F5**. After the simulation completes, right-click on the Function element and then left-click on **Time History Result…** in the context menu to view a time history plot of the result:



Note: Your result plot may look different, since this represents one random realization of the system.

What are we looking at here? This is actually the status of the Component over the course of the simulation. 0 indicates that the component is operating, and 2 indicates that the component has failed due to unmet Internal Requirements (in this case, the occurrence of the Simple Poisson failure mode).

Before thinking about this result plot further, it's important to first discuss how GoldSim moves the simulation through time. GoldSim "updates" the model as follows:

1. The model is updated (all the elements in the model are recalculated) at the timesteps specified by the user. In this case, this means that element values are recalculated at 0 years, 1 year, 2 years, 3 years, 4 years and 5 years (as we have five timesteps). Values for time history plots are recorded at these user specified timesteps.
2. GoldSim also updates the model upon the occurrence of certain kinds of "events". The failure of a reliability element is such an event. Hence, in this case, if the Component failed at 1.57 years, GoldSim would insert a new timestep at this point and update the model.

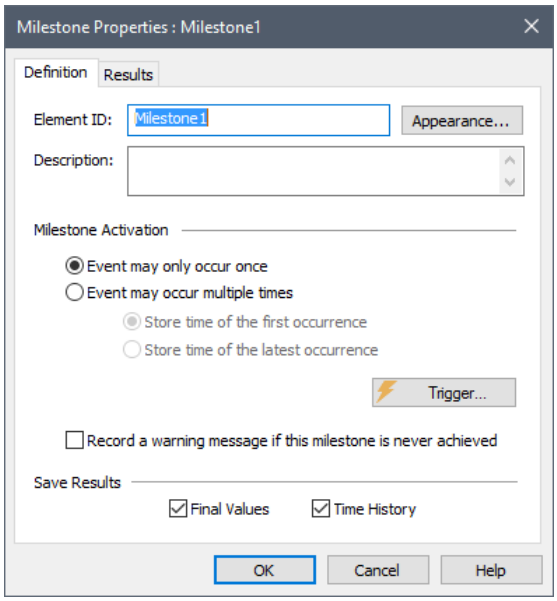
Note, however, that although the elements in the model are updated when such an event occurred, values for time history plots are only recorded at the user specified timesteps (in this case, 0, 1, 2, 3 ,4 and 5 years).

Given this information, we now know that the result shown above indicates that in this particular random realization (again, yours may be different), the Component failed sometime between 1 and 2 years. In the next step, we will determine exactly when the Component failed.

Step 4: Determining the Time of Failure Using a Milestone Element

In order to determine exactly when the Component failed, we will add a Milestone element to the model. A Milestone element is an element that is part of the basic GoldSim framework. The element is quite simple. In its simplest implementation, it is triggered by an event, and simply records the time when it was triggered.

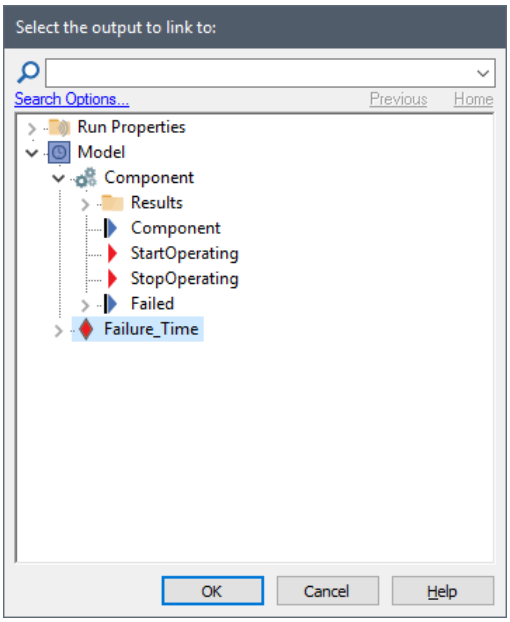
To insert the Milestone element, we need to return to Edit Mode by pressing **F4**. We can then insert the element by right-clicking in the graphics pane, and selecting **Insert Element|Events|Milestone** from the context menu. A Milestone element will be inserted and its property dialog will be displayed:



Change the element's name to "Failure_Time". Since the Component can only fail once in this simple model, the default Milestone Activation settings do not need to be changed.

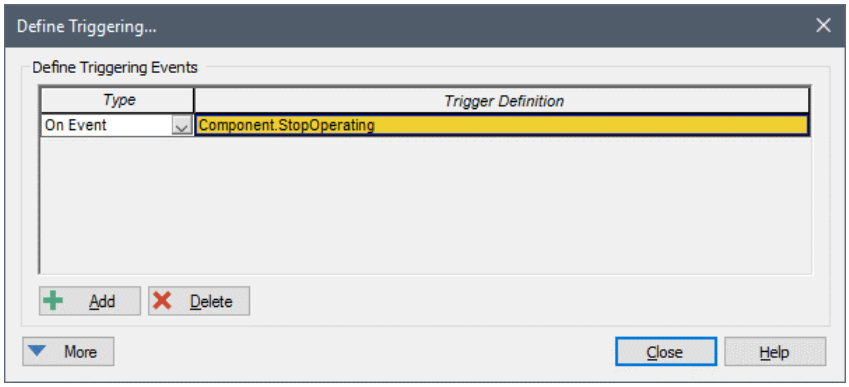
We now need to trigger the Milestone when the Component stops operating. This can be done as follows:

1. Press the **Trigger…** button.
2. Press the **Add** button in the Trigger dialog to add a trigger.
3. The default Type will be On Event. This can be left unchanged. To specify the **Trigger Definition**, right-click in the field and select **Insert Link** from the context menu.
4. Expand the Component element by clicking on the > sign. The following will be shown:



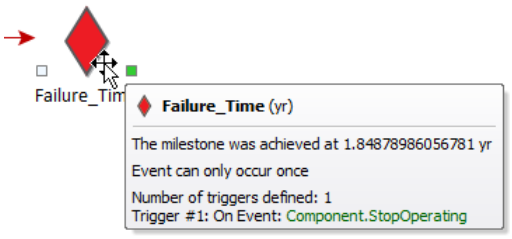
5. Double-click on the StopOperating output. This is an event that is emitted by the element when the Component stops operating.

At the end of this process, the Trigger dialog should look like this:



Close the Trigger dialog and the Milestone's Properties dialog and run the model again (**F5**).

After you rerun the model, you can mouse over the Milestone element to see the time that the Component actually failed. In the example shown below, you can see that it failed at approximately 1.85 years:



Note: As pointed out previously, your result will be different, since this represents one random realization of the system

Step 5: Increasing the Level of Time Discretization

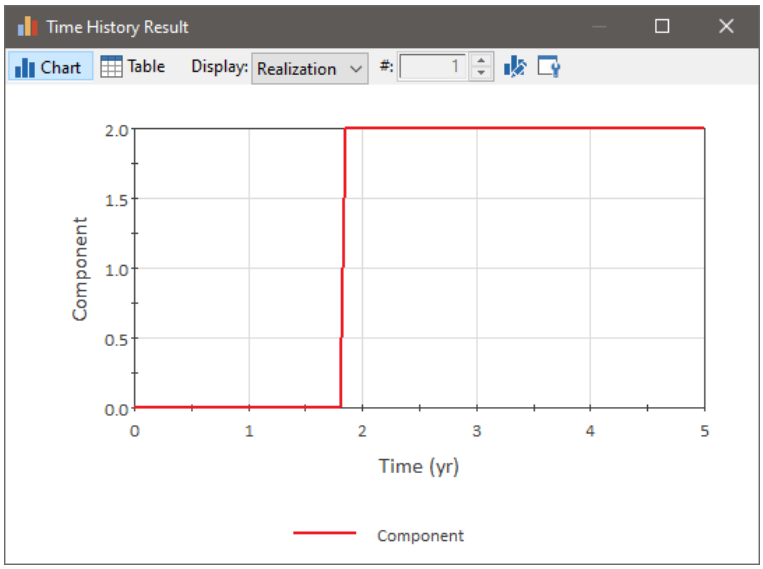
Let's add some additional timesteps and rerun the model to see how the time history plot changes.

To do this, you'll need to return to Edit Mode by pressing **F4**. Then, open the Simulation Settings dialog (by pressing **F2**) and change the **Basic Step** length to 0.05 yr.

We can now close the Simulation Settings dialog and rerun the model (**F5**).

If you mouse over the Milestone element, you will notice that the time of its occurrence hasn't changed from when you looked at them the last time your ran the model (since by default, random numbers generated to produce the events are repeatable).

However, the time history plot will change significantly. We can see this by right-clicking on the Function element and then left-clicking on **Time History Result…** in the context menu to view a time history plot of the result:



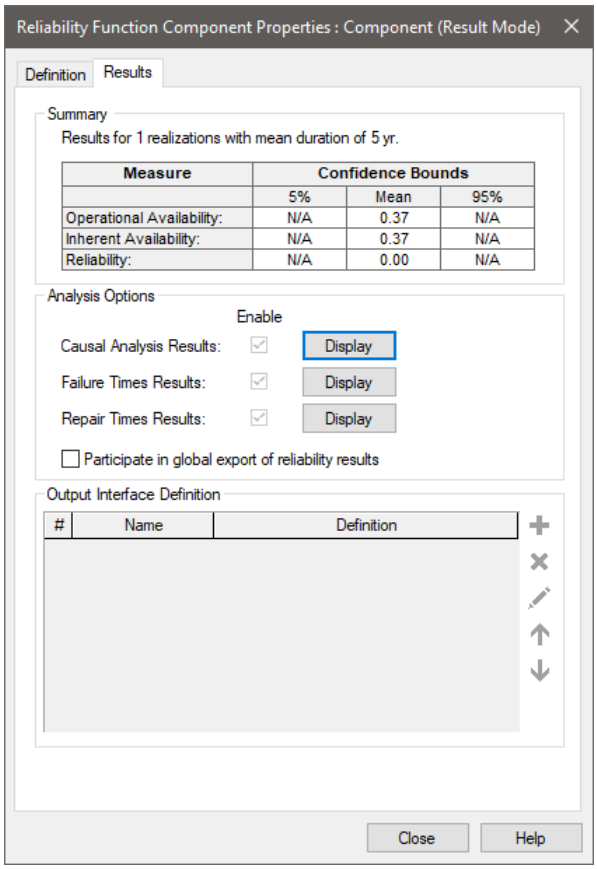
Notice how the change in the component's status from 0 to 2 is now seen to indeed occur around 1.85 years due to the additional timesteps.

Note: As pointed out previously, your result plot will be different, since this represents one random realization of the system.

Step 6: Computing Reliability and Availability

Although simple time histories of failure for a single realization may be of some interest, what we are really interested in doing is computing the reliability and availability of the Component.

We can view these results by opening the Component's property dialog and viewing the **Results** tab:



Note: As pointed out previously, your result will be different, since this represents one random realization of the system

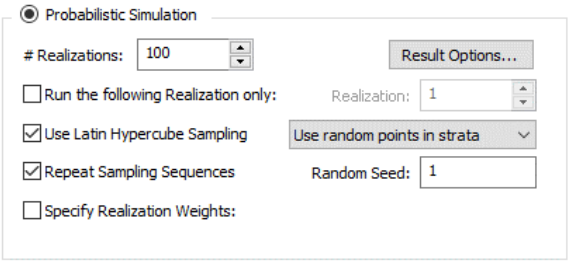
The results shown above show an Availability of 0.37 and a Reliability of 0. What does this mean? The problem is that these metrics cannot be accurately determined by running a single realization of the model. The Availability is simply being computed here as the time the Component was operating (in this case about 1.85 years) divided by the total simulation time (5 years). The Reliability is zero because the Component failed in this particular realization.

We cannot draw accurate statistical conclusions with a sample size of one. In order to discern the expected behavior of the system, and the boundaries of that behavior, we need to either run multiple realizations or, if the system was being repaired, run a single realization for a long time so that it models multiple failure and repair cycles. In the next step, we will do the former.

Step 7: Running Multiple Realizations of a Reliability Model

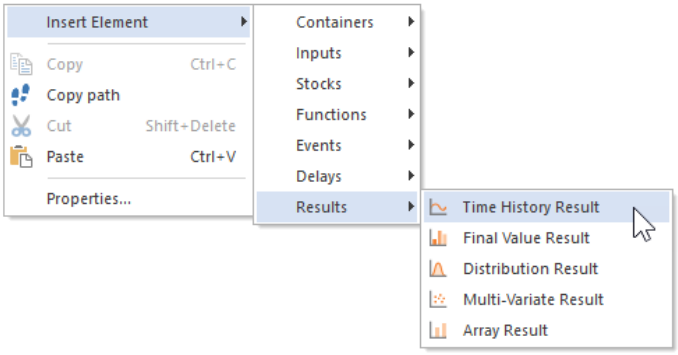
In order to discern the expected reliability and availability of the system, and the range of that behavior, we will run multiple realizations of the system using Monte Carlo simulation.

For this simple simulation, 100 realizations will provide a good sample to work with. To change the number of realizations, return to Edit Mode (press **F4**), and open the Simulation Settings dialog (press **F2**). Click on the **Monte Carlo** tab, and make sure the **Probabilistic Simulation** radio button is selected. Then change the **# Realizations** to 100:

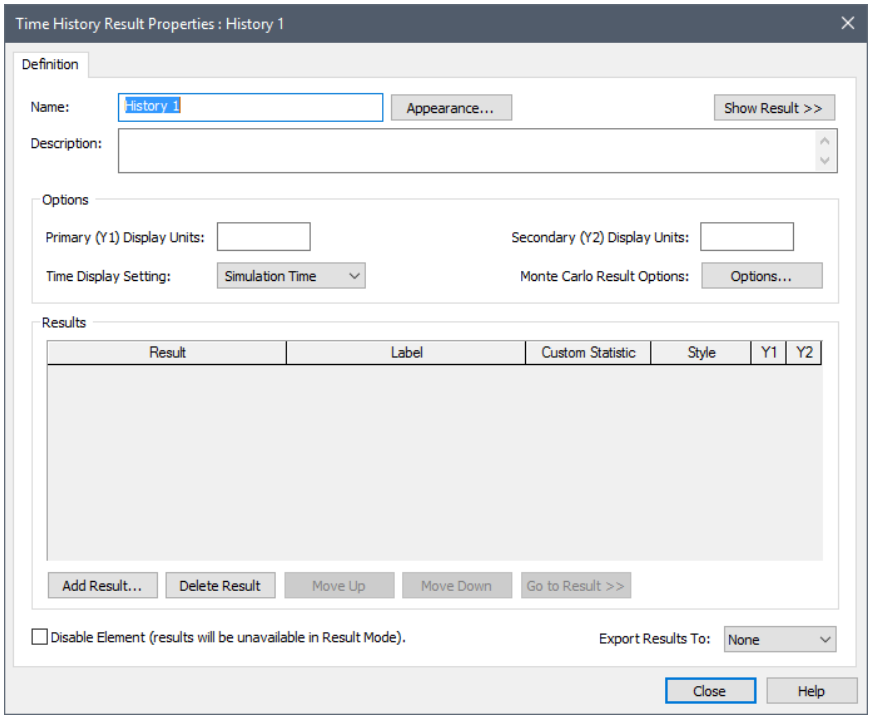


Close the Simulation Settings dialog.

When running multiple realizations, in order to view time history results, you must connect the output of interest to a Time History Result element. To do so, right-click in the graphics pane and insert a Time History Result element:



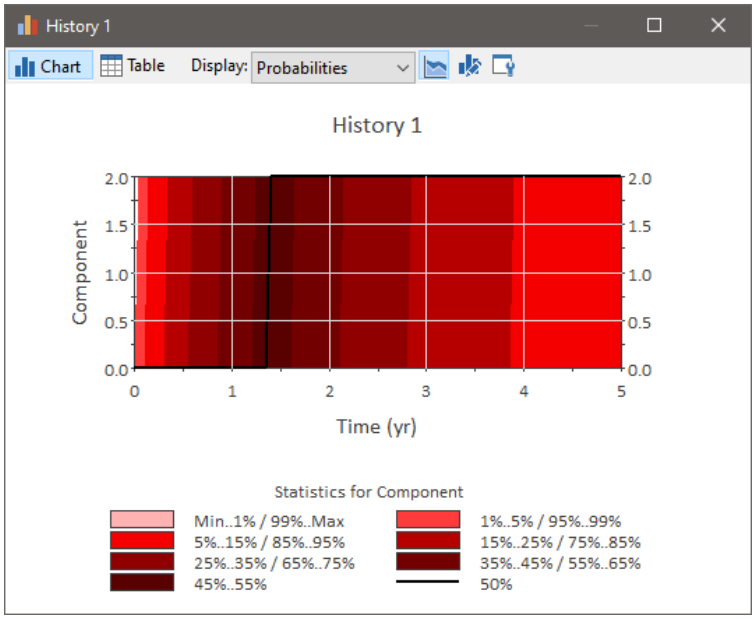
The following dialog will appear:



Press the **Add Result…** button and select Component. Then close the dialog and run the model (by pressing **F5**).

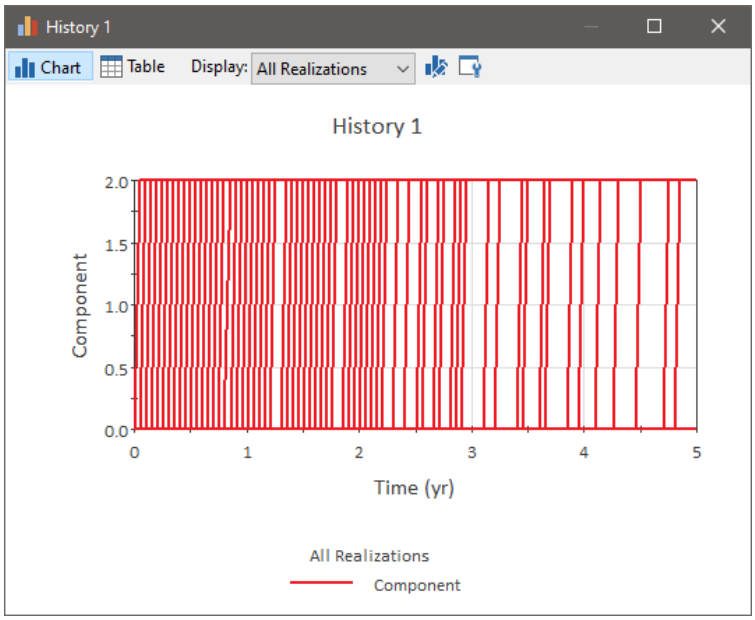
Step 8: Viewing Monte Carlo Results for a Reliability Model

After running our simple Monte Carlo simulation of the Component, we can view the result in a variety of ways. Let's start by viewing a time history plot. We can do so by double-clicking on the Time History Result element:



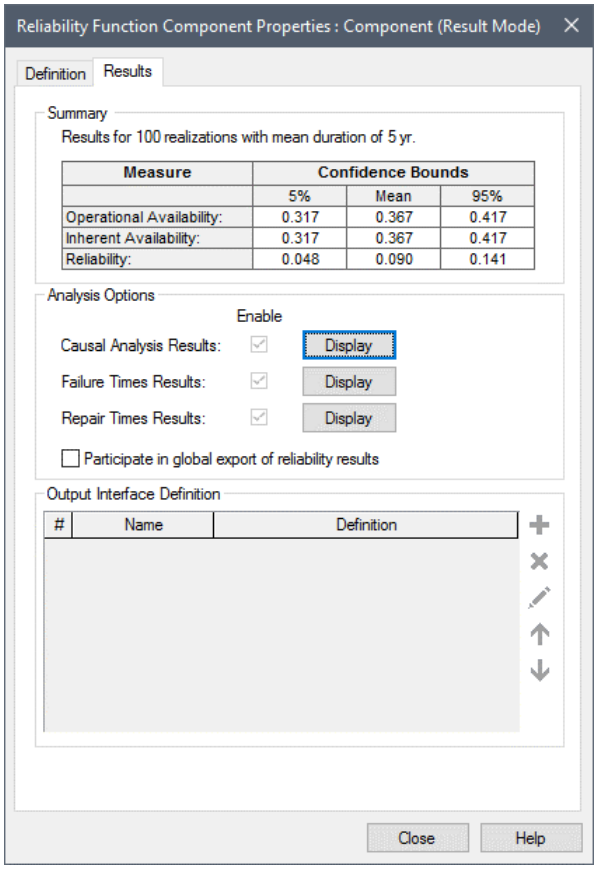
When you initially view a time history plot of multiple realizations, GoldSim displays a probability history view, showing you the median (black line), and percentiles on the Component's status over time. In this example, the various histories are simply vertical lines, hence the percentile boundaries are vertical lines.

Things become a bit clearer if we switch from displaying Probabilities" to displaying "All Realizations" (by selecting this option from the **Display** drop-list at the top fo the chart)::



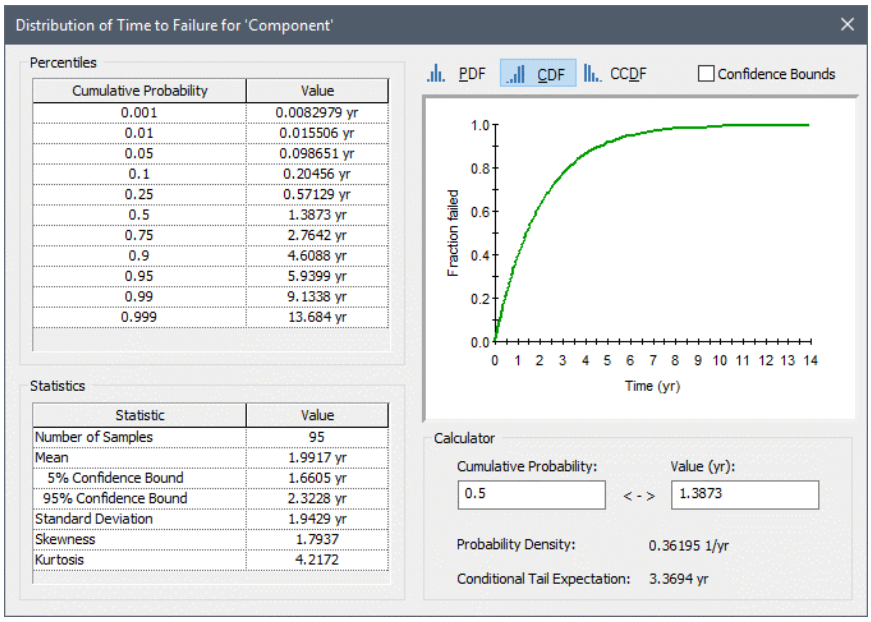
Each line represents a different realization (with the Component failing at different times). Statistical results on the expected behaviour of a system are produced by combining the results of these realizations.

We can see these statistical results by viewing the **Results** tab of the Component element:



Notice that the reliability value is now nonzero (i.e., there is a small, but nonzero probability that the Component will operate for the full five year period). GoldSim displays Availability and Reliability with confidence bounds, Note that the range between the confidence bounds would shrink if we increased the number of realizations.

Now click on the **Display** button to the right of **Failure Times Results** label. This will display a distribution of the time of failure for the Component:

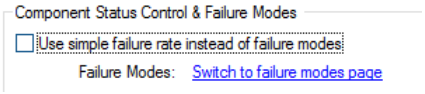


Step 9: Editing Failure Modes and Adding Automatic Repair

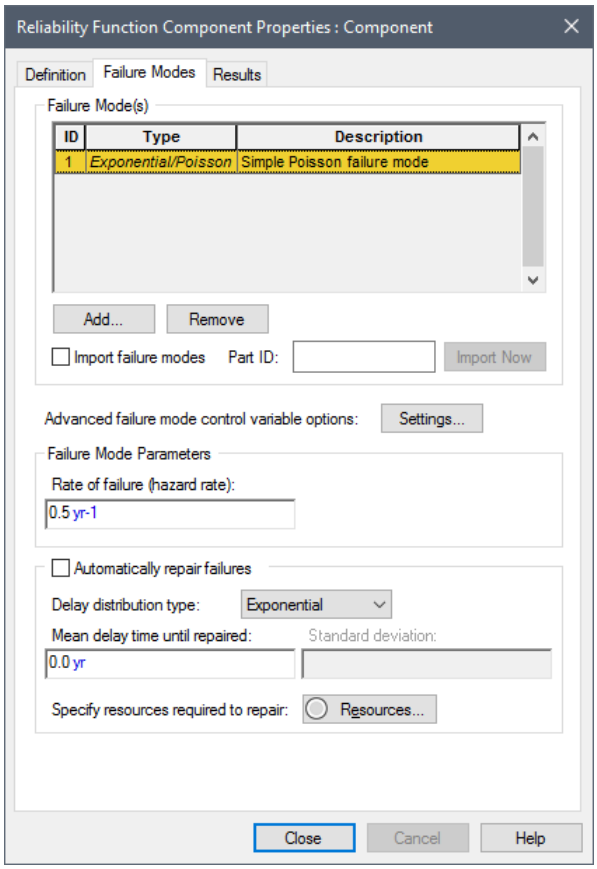
Let's add some additional detail to the Component's behaviour by adding failure modes and repair.

By default, the Function element allows you to specify a simple exponential failure mode that is not repaired. However, in many cases, the component's behaviour cannot be accurately modeled in such a simple way. Moreover, many high value systems do not "fail" permanently – they are repaired and returned to service.

To demonstrate this capability, let's return to Edit Mode (**F4**) and edit the failure modes for the Component. To do so, double-click on the Component element and clear the **Use simple failure rate instead of advanced failure modes** checkbox in the element's property dialog:

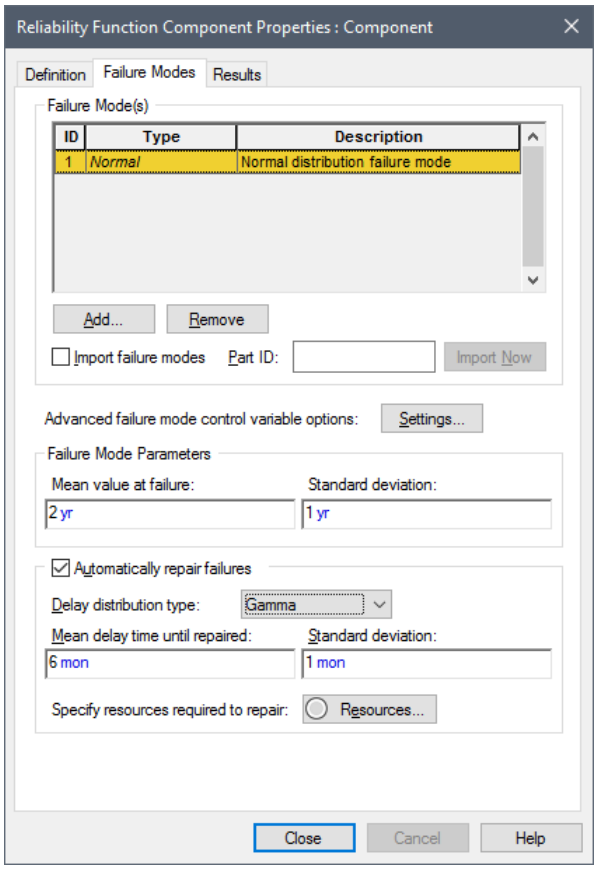


You'll notice that a new tab, called **Failure Modes** will appear. If you click on this tab you'll see the following dialog:



You'll notice that the initial exponential failure mode has been transferred over automatically by GoldSim. Let's delete this failure mode (press the **Remove** button), and add a Normal failure mode (by pressing the **Add** button and selecting "Normal"). Specify a mean of 2 yr and a standard deviation of 1 yr.

You'll also notice that there is an **Automatically repair failures** option. Check that option, and specify a repair with a Gamma delay distribution, a mean time to repair of 6 months and a standard deviation of one month (month is abbreviated as "mon" in GoldSim). The dialog should then look like this:

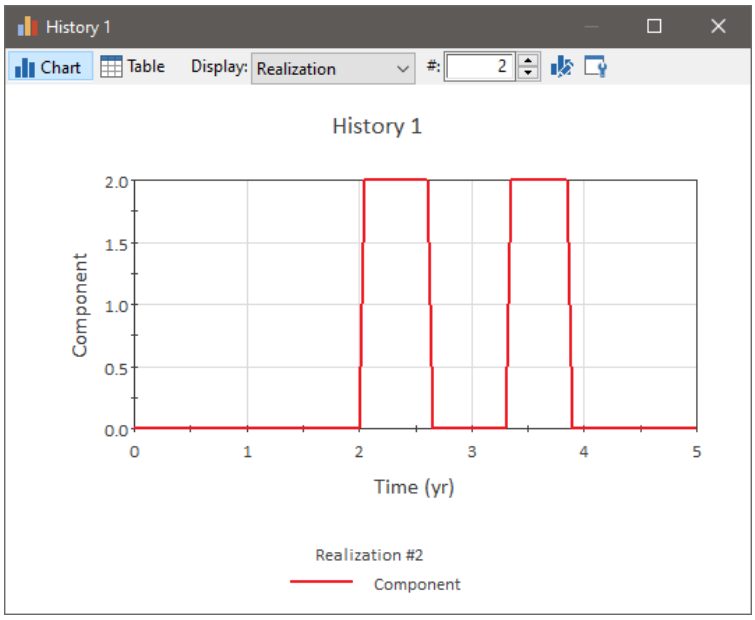


If we wanted to, we could add additional failure modes. For the sake of simplicity, however, we'll stick with one for now. Press **Close** to close this dialog.

Delete the Milestone element (as it will generate warnings since it will now be triggered multiple times each realization due to the repair of the failure mode), and re-run the model (**F5**).

Let's view a time history plot of the status of the Component by double-clicking on the Time History Result element. From the **Display** drop-list at the top of the chart, select "Realization". This shows one realization at a time. You can toggle through realizations using the **Realization** spin control at the top of the window.

A typical result should look something like this:



As can be seen, whenever the Component fails, it is repaired about 6 months later. If we look at the **Results** tab for the Component, we will see that the Availability is relatively high (due to the repairs):

Summary

Results for 100 realizations with mean duration of 5 yr.

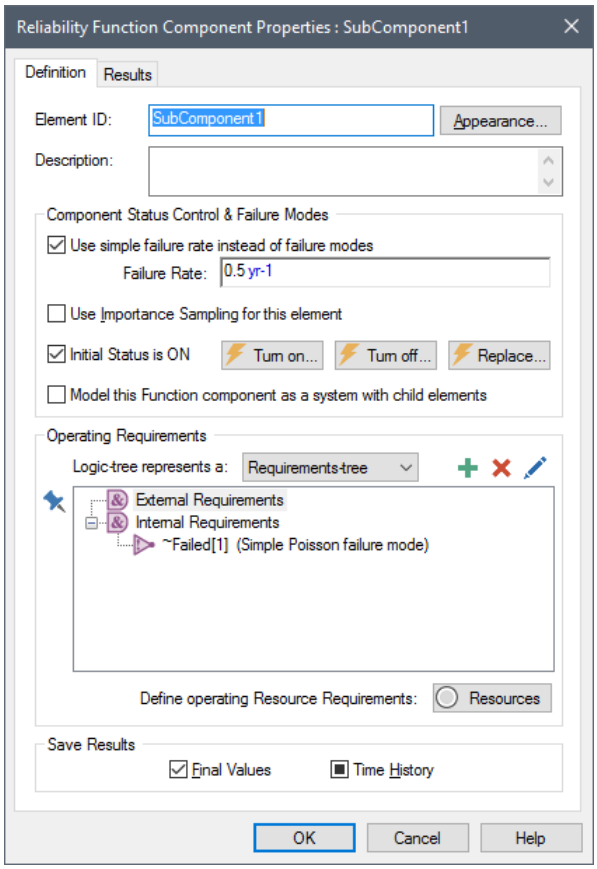
Measure	Confidence Bounds		
	5%	Mean	95%
Operational Availability:	0.820	0.831	0.841
Inherent Availability:	0.820	0.831	0.841
Reliability:	0.000	0.000	0.000

Step 10: Adding Hierarchy (Sub-Components) to a Reliability Model

Modeling the Component as a single object is the simplest way to represent the system. Since the Component actually consists of sub-components (each of which can fail in its own way), however, we may decide to model the system hierarchically. Because GoldSim and the Reliability Module are hierarchical by nature, we can quickly transform our simple single element model into a system with a number of subcomponents.

To convert the model of the Component into a system with child (sub) elements, return to Edit Mode (**F4**), open the Component's dialog and check the option to **Model this Function component as a system with child elements**. Then close the dialog. You will notice that there is a slight change in the appearance of the Component element. In particular, a small red triangle now appears in the upper left-hand corner of the element (the Component is now equivalent to a Container in GoldSim).

Clicking on the triangle takes you into a new layer of the model *within the Component*. After doing so, let's add three new Function elements (named SubComponent1, SubComponent2, and SubComponent3). Each one will be defined identically, with a failure rate of 0.5 yr-1:



Of course, we could have defined multiple failure modes and automatic repairs for each subcomponent, but for simplicity, we will use a simple failure rate here.

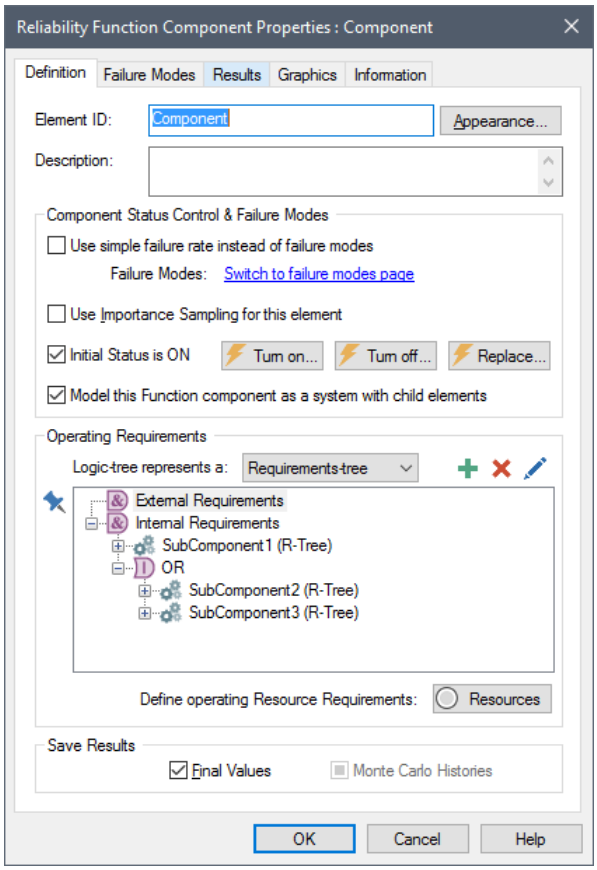
After creating these three new elements, move one level back up in the hierarchy, and open the Component element again. Go to the **Failure Mode** tab and delete the failure mode. That is, instead of specifying a failure mode for the Component, we want to specify that the Component fails when its subcomponents fail. We do this using the Operating Requirements section of the Component's dialog.

Let's assume that in order for the Component to operate, SubComponent1 must be operating and either SubComponent2 or SubComponent3 must be operating (SubComponent2 and SubComponent3 are redundant). We can do this by building an Internal Requirements tree for the Component as follows:

1. Left-click on Internal Requirements (which is an AND node).
2. Click on the "+" sign above the tree. This will display a menu to add a new node under the Internal Requirements.
3. Select "RL Component" from this menu. This will display a browser. Find SubComponent1 and double-click on it.
4. Click on the "+" sign again and Select "OR-Gate" from the menu.
5. The Or-Gate will be selected. Click on the "+" sign again and Select "RL Component" from the menu. This will display a browser. Find SubComponent2 and double-click on it.

6. Click on the "+" sign again and Select "RL Component" from the menu. This will display a browser. Find SubComponent3 and double-click on it.

At the end of this process, the dialog for the Component should look like this:



This indicates that the Component does not fail directly. Rather, it remains operable as long as SubCompenent1 has not failed and either Subcomponent2 or Subcomponent3 has not failed (i.e., it fails if SubComponent1 fails or both SubComponent2 and SubComponent3 fail).

If you wish, you can rerun the model now to see how this new representation of the system impacts the reliability.

Where Do I Go From Here?

This chapter was intended to provide an introduction to the features and capabilities of the Reliability Module.

The next four chapters (Chapters 3, 4, 5 and 6) provide detailed explanations of the features introduced here. The final chapter (Chapter 7) discusses a number of example models that illustrate how to represent a variety of systems. These example models are automatically installed with GoldSim.

You may want to start by jumping to the final chapter and experimenting with some of the example models. You will then likely want to refer back to the four chapters describing the details of the Reliability Module to fully understand how the examples were constructed.

Chapter 3: The Reliability Elements

When you have two engines, you have two engines that can fall to bits. When you have four, you have four that can fall to bits. The less engines you have, the safer you are.

Chief Engineer for large airplane manufacturer (replying to a pilot association's complaint about the dangers of flying two-engine planes across the Pacific)

Chapter Overview

This chapter provides details on the purpose and use of the Function and the Action reliability elements.

In this Chapter

This chapter discusses the following topics:

- The Difference Between the Function and the Action Elements
- Overview of the Function Element
- Overview of the Action Element
- The Common Inputs and Features of the Reliability Elements
- The Common Outputs and Locally Available Properties of the Reliability Elements
- Defining Operating Requirements for Reliability Elements Using Logic Trees
- Inputs, Outputs and Features Specific to the Action Element

The Difference Between the Function and the Action Elements

The Reliability Module provides two elements: the Function element and the Action element.

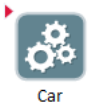
Function elements are used to represent services that are carried out over a period of time such as an air-conditioning system, a battery, or a back-up generator. The Function element provides outputs that at any given time in a simulation inform other elements in the model whether or not it is functioning correctly. This allows for dependency relationships to be easily defined, for example a *radio* component might require a *power-supply* component to be operating.

Action elements are very similar to Function elements, but have several additional features to facilitate representation of activities or processes that are carried out discretely (as opposed to continuously) such as when a door latches closed, a switch opens or closes, an engine starts, or a message is delivered. The Action component waits for a triggering input to tell it to carry out its action. If its action succeeds, the element emits an 'ActionOK' event, and if it fails the element emits an 'ActionFailed' event.

Overview of the Function Element

Function elements are used to represent services that are carried out over a period of time.

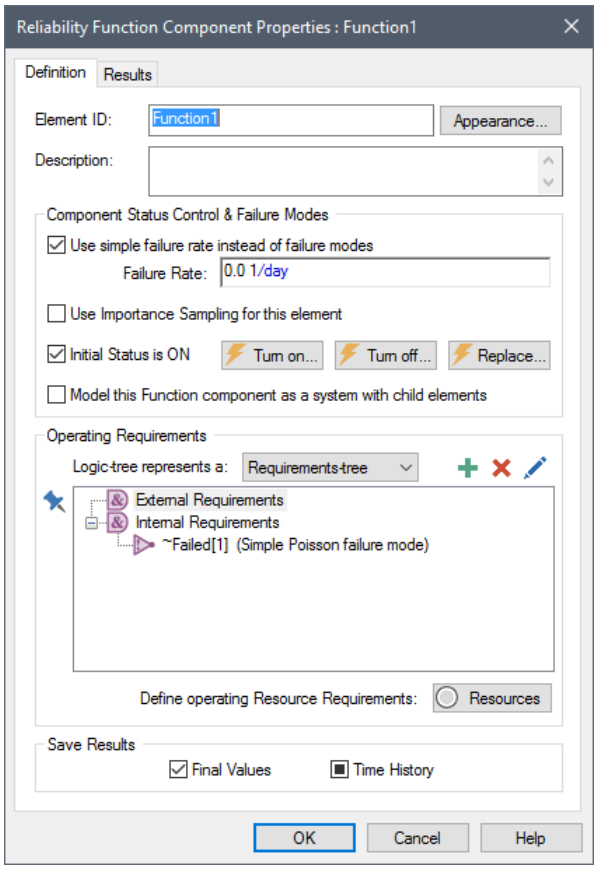
The default icon for the Function element looks like this:



The Function element has several outputs, but the key output is its Status (the primary output of the element). The Status is an integer that takes on one of a number of values. Typically, we are most interested in whether the component is operating (Status = 0). In order for a Function element to be operating, it must be turned "On". It will cease operating if it is 1) turned "Off"; 2) it fails for some reason; 3) it is being maintained; 4) its specified operating requirements are not met; or 5) it is part of a subsystem and its parent stops operating. Each of these states is indicated by a unique value for the Status.

Read more: Common Reliability Element Outputs (page 60).

The Properties dialog for the Function element looks like this:



The key parts of the dialog are summarized below.

The "Component Status Control & Failure Modes" portion of the dialog is where the primary properties of the component being modeled are specified, including simple failure rates, the initial status of the component, and triggers for turning the component on and off and for replacing it:

Use simple failure rate instead of failure modes. When this checkbox is selected, the **Failure Rate** specified in the text box below is used as the only failure mode (this assumes an exponential/Poisson failure). When the checkbox is cleared, a **Failure Modes** tab will appear, and you can use GoldSim's advanced failure modes instead of a failure rate.

Read more: Failure Rates and Failure Modes (page 56).

Use Importance Sampling for this element: Checking this option enables a modified sampling algorithm that allows GoldSim to better represent low-probability (i.e., rare) failure events without requiring a large number of realizations. Note, however, that this advanced feature must be used with caution.

Read more: Using Importance Sampling for Reliability Elements (page 57).

Initial Status is On. If this box is checked, the component is assumed to be "On" at the beginning of the simulation.

Turn On, Turn Off and Replace Triggers. These buttons provide access to triggers that allow you to turn the component On or Off or replace it. Replacing the component "resets the clock" for all failure modes.

Read more: Turning Components On and Off in the Reliability Module (page 90).

Model this Function component as a system with Child Elements. This checkbox allows you to turn the element into a Container and place other GoldSim elements inside this element. Note that when you do so, a new tab will appear (**Graphics**). This tab is common to all Containers in GoldSim.

Read more: Modeling a Reliability Element as a System with Child Elements (page 58).

The "Operating Requirements" portion of the dialog allows you to define (in the form of a Requirements-tree or Fault-tree) the conditions necessary for the component to operate. You can also specify Resource Requirements necessary for the component to operate.

Read more: Defining Operating Requirements for Reliability Elements Using Logic Trees (page 64); Modeling Resources in the Reliability Module (page 121).

All Reliability elements have a **Results** tab. After a simulation has been run, the **Results** Tab can be used to access result statistics and the unique causality trees and root causes which resulted in failure of the element.

Read more: Chapter 6: Displaying Reliability Results (page 129).

Overview of the Action Element

Action elements are very similar to Function elements, but have several additional features to facilitate representation of activities or processes that are carried out discretely (as opposed to continuously). The Action component waits for a triggering input to tell it to carry out its action. If its action succeeds, the element emits an 'ActionOK' event, and if it fails the element emits an 'ActionFailed' event.

The default icon for the Action element looks like this:

The Action element has several outputs, but there are three key outputs that are worth mentioning here.

The primary output of the element is the Status. The Status is an integer that takes on one of a number of values. Typically, we are most interested in whether the component is operating (Status = 0). In order for an Action element to be operating, it must be turned "On". It will cease operating if it is 1) turned "Off"; 2) it fails for some reason; 3) it is being maintained; 4) its specified operating requirements are not met; or 5) it is part of a subsystem and its parent stops operating. Each of these states is indicated by a unique value for the Status.

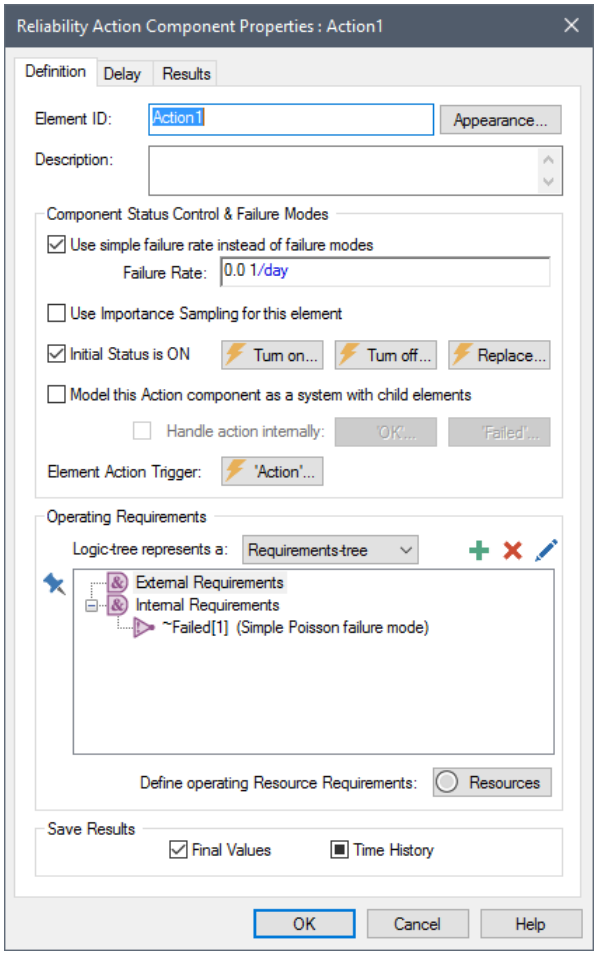
When the component is triggered to act, it can only complete its action if it is operating. If it completes its action, the element outputs a discrete event signal (called ActionOK). If it is not operating when it is triggered to act, the action is considered to have failed, and the element outputs a discrete event signal (called ActionFailed).

Read more: Common Reliability Element Outputs (page 60); Outputs Available Only for Action Elements (page 77).

Note: Even if the component is operating, you can optionally specify a failure mode for the component that determines whether or not it will complete the action when triggered.

Read more: Failure Modes Available Only for Action Elements (page 99).

The properties dialog for the Function element looks like this:



The key parts of the dialog are summarized below.

The "Component Status Control & Failure Modes" portion of the dialog is where the primary properties of the component being modeled are specified, including simple failure rates, the initial status of the components, and triggers for turning the component on and off and for replacing it:

Use simple failure rate instead of failure modes. When this checkbox is selected, the **Failure Rate** specified in the text box below is used as the only failure mode (this assumes an exponential/Poisson failure). When the checkbox is cleared, a **Failure Modes** tab will appear, and you can use GoldSim's advanced failure modes instead of a failure rate.

Read more: Failure Rates and Failure Modes (page 56).

Use Importance Sampling for this element: Checking this option enables a modified sampling algorithm that allows GoldSim to better represent low-probability (i.e., rare) failure events without requiring a large number of

realizations. Note, however, that this advanced feature must be used with caution.

Read more: Using Importance Sampling for Reliability Elements (page 57).

Initial Status is On. If this box is checked, the component is assumed to be "On" at the beginning of the simulation.

Turn On, Turn Off and Replace Triggers. These buttons provide access to triggers that allow you to turn the component On or Off or to replace it. Replacing the component "resets the clock" for all failure modes.

Read more: Turning Components On and Off in the Reliability Module (page 90).

Model this Function component as a system with Child Elements. This checkbox allows you to turn the element into a Container and place other GoldSim elements inside this element. Note that when you do so, a new tab will appear (**Graphics**). This tab is common to all Containers in GoldSim.

Read more: Modeling a Reliability Element as a System with Child Elements (page 58).

Handle action internally. This checkbox controls how Actions that are defined as systems handle action triggers. If the box is checked, the triggers are automatically routed to internal Action components. If this box is checked the OK and Failed triggers become available (for handling responses from internal elements).

Read more: Handling Actions Internally (page 125).

Element Action Trigger. This provides access to the trigger(s) that cause the action to be carried out.

Read more: Triggering the Action Element (page 74).

The "Operating Requirements" portion of the dialog allows you to define (in the form of a Requirements-tree or Fault-tree) the conditions necessary for the component to operate. You can also specify Resource Requirements necessary for the component to operate.

Read more: Defining Operating Requirements for Reliability Elements Using Logic Trees (page 64); Modeling Resources in the Reliability Module (page 121).

The Action element also has a **Delay** tab. By default, an Action, when triggered, is carried out instantaneously. This tab, however, allows you to specify a duration for the action.

Read more: The Delay Tab of the Action Element (page 78).

All Reliability elements have a **Results** tab. After a simulation has been run, the **Results** Tab can be used to access result statistics and the unique causality trees and root causes which resulted in failure of the element.

Read more: Chapter 6: Displaying Reliability Results (page 129).

The Common Inputs and Features of the Reliability Elements

The two reliability elements (Functions and Actions) have a number of common inputs and features. These are discussed in detail in the following sections.

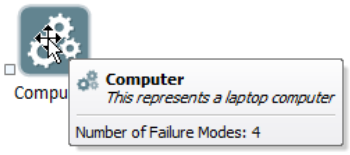
Features the Reliability Elements Share with All GoldSim Elements

Like all GoldSim elements, Function elements and Action elements have an **Element ID**, **Description**, **Appearance** button, and two **Save Results** checkboxes.

The default **Element ID** (i.e., name) of an element is the element type followed by a number (e.g., Function3, Action1). After creating a new reliability element, you should change the element name to be more meaningful.

Note: Element names can only include letters, numbers, and the underscore ("_") character. They cannot include spaces and must begin with a letter. In addition, some names (e.g., sin, cos, min) are reserved for use as functions and cannot be used as an element name.

You can enter a **Description** for the element below the ID. Note that this description appears in the tool-tip displayed for the element in the graphics pane:



A tool-tip is displayed when you hold the cursor over the element. The tool-tip includes the element's description.

You can access a menu for changing the graphical appearance of an element by pressing the **Appearance…** button.

The two checkboxes in the **Save Results** section allow you to choose whether to save **Final Values** (the values at the end of each realization in the simulation) and/or **Time History** (the value at selected timesteps throughout the simulation). Typically, these are specified by separately setting flags for each element.

Note: The **Final Values** checkbox always controls whether Final Value results are saved. However, the **Time History** checkbox can be overridden. In particular, it is always applied for single realization runs, but is overridden for multiple realization runs. In particular, Time History results for multiple realization runs are only saved for outputs that are connected to Result elements.

With a few exceptions, almost all GoldSim property dialogs include one or more input fields, which allow you to define inputs to the element.

You can specify the contents of an input field in a properties dialog in one of three ways:

- You type in a number;
- You create a link to another element by entering the name of an output of that element; or
- You enter a mathematical expression, such as 3*log(250). Note that the expression itself can incorporate links (the names of element outputs), such as A*log(B), where A and B are output names.

It is important to note that the value in the input field does not need to be constant throughout the simulation, and can change through time. For example, an expression might reference the current value of simulation time, or a link

could be created to an element whose outputs varied during the course of the simulation.

Note: Editing an element's name, description and appearance, as well as creating links and entering and editing expressions in input fields is discussed in detail in Chapter 3 of the **GoldSim User's Guide**.

Failure Rates and Failure Modes

The default failure mode for both types of reliability elements is an exponential/Poisson failure mode that is never repaired.

The **Failure Rate** (also known as the hazard rate) represents the mean failure rate (the inverse of the mean time to failure), and has dimensions of inverse time. Failure is assumed to be a Poisson process (which implies that if the rate is constant, the time between events is exponentially distributed).

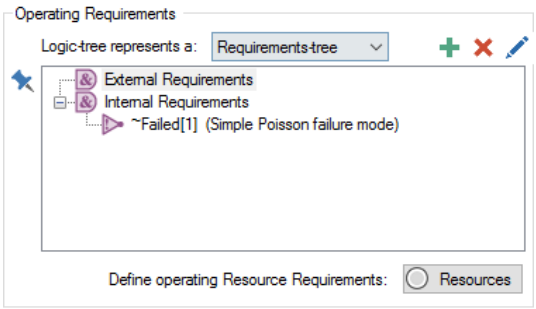
Note: The mathematics of exponential/Poisson failure are described in detail in Appendix A.

Note that the Failure Rate does not have to be constant over time. You can enter this input as an expression (or a link from another element) that varies with time.

Failures modeled in this way are computed with respect to the time since the simulation started. If you wish to compute failures based on the operating time, or perhaps some other indirect metric (e.g., cumulative mileage), you must activate the advanced failure mode feature (by clearing the **Use simple failure rate instead of failure modes** checkbox). When you do so, a **Failure Modes** tab is added to the element, allowing advanced failure mode options to be accessed.

Warning: The default exponential failure mode is a very simple way to model failure (and therefore should be used with caution). An unrepaired, exponential failure (with a constant hazard rate) may be appropriate for random (and typically externally-generated) failures, but is not likely to be appropriate for failure due to processes such as wear and tear. As a result, GoldSim's advanced failure mode features should be used for most components.

By default, failures are assumed to be fatal to the component. That is, if a particular failure mode occurs, the component itself fails and is no longer operative. As a result, you will note that all failure modes are displayed as nodes (in the form of "Internal Requirements") in the element's Operating Requirements logic-tree:



Read more: Operating Requirements (page 59).

Advanced failure mode options include the ability to add multiple failure modes (using a variety of failure distributions), the ability to create non-fatal failure and coupled failure modes, and the ability to simulate repair and preventive maintenance.

Read more: Defining Failure Modes (page 92).

Using Importance Sampling for Reliability Elements

For risk analyses, it is frequently necessary to evaluate the consequences of low-probability, high-consequence failures (i.e., failures that occur with a very low frequency, but have a significant impact on the system). Because the models for such systems are often complex (and hence need significant computer time to simulate), it can be difficult to use the conventional Monte Carlo approach to evaluate these low-probability, high-consequence failures, as this may require excessive numbers of realizations.

To facilitate these type of analyses, GoldSim allows you to utilize an ***importance sampling*** algorithm to modify the conventional Monte Carlo approach so that high-consequence, low-probability failures are sampled with an enhanced frequency. That is, importance sampling serves to increase the rate of occurrence of the failure. During the analysis of the results that are generated, the biasing effects of the importance sampling are reversed. The result is high-resolution development of the high-consequence, low-probability "tails" of the consequences (resulting from low-probability failures), without paying a high computational price.

Importance sampling for Reliability elements is specified by selecting the checkbox for **Use Importance Sampling for this element**. The algorithm that is used is discussed in detail in Appendix B of the **GoldSim User's Guide**.

Four points regarding importance sampling of failures for Reliability elements should be noted:

- Importance sampling is only applied to the first occurrence of the failure. The increased sampling frequency is not applied to subsequent occurrences within the same realization.
- Importance sampling of Reliability elements should only be used to represent failures that are *rare*. As used here, "rare" indicates a failure that would not be represented adequately without enhanced sampling. As a general rule of thumb, a failure event will be adequately sampled if the product of the number of realizations and the expected number of failures over the course of a single realization is at least 10 (this would indicate that the failure would occur on average at least 10 times if that many realizations were executed). For example, if the rate of failure was once every thousand years, and the simulation was run for 10 years, one could expect 0.01 failures per realization. If 100 realizations were run, the total expected number of failures (over all realizations) would therefore be 1. This is a rare failure and importance sampling should be applied (or more realizations should be run).
- There is a limit to the effectiveness of importance sampling for extremely rare failures, such that in some cases, it may become necessary to increase the number of realizations in order to effectively represent the failure. In particular, the degree of biasing for low probability failures that GoldSim can provide is (*at most*) equal to the number of realizations. For example, if the failure rate was once every 100,000 hours, and the simulation was run for 10 hours, one could expect 1e-4 failures per realization. If 100 realizations were run, the

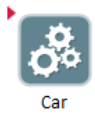
total expected number of failures (over all realizations) would therefore be 0.01. This is a rare failure and importance sampling should be applied. However, importance sampling could only increase the total number of failures (over all realizations) by a factor of 100 (the number of realizations) to 1 (which would still provide inadequate representation). If 1000 realizations were run, the total expected number of failures (over all realizations) without importance sampling would be 0.1, but importance sampling would improve it by a factor of 1000.

- You should use importance sampling sparingly (i.e., only for those elements that really need it). This is because, as stated above, the degree of biasing for low probability failures that GoldSim can provide is *at most* equal to the number of realizations, and the actual biasing provided decreases with the number of elements for which importance sampling is applied.

Modeling a Reliability Element as a System with Child Elements

In some cases, it may be appropriate to model a component as a system of sub-components. For example, you might want to model a computer as a system consisting of sub-components such as a power supply, a motherboard, a DVD drive, and a monitor. You could specify that in order for the computer to operate, the power supply, motherboard and monitor must all operate. You could further specify that the DVD drive could be inoperable without causing the computer to be inoperable.

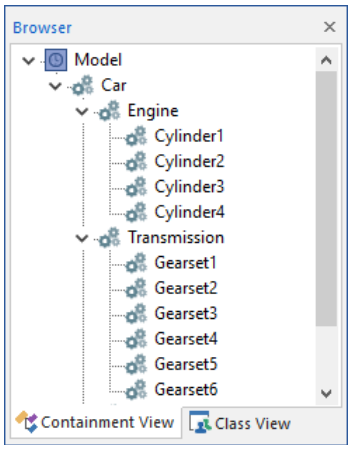
You can turn a reliability element into a system by checking the **Model this element as a system with child elements** box. When you do so, there is a slight change in the appearance of the reliability element:



A small red triangle appears in the upper left-hand corner of the element, and clicking on the triangle takes you into a new layer of the model *within the component*. If you are familiar with GoldSim, you will recognize this plus sign as the same plus sign that normally appears on the GoldSim Container elements. In fact, when we convert a reliability element into a system, the element retains all of its properties, but now also functions as a (localized) Container.

Note: Containers are discussed in Chapter 3 and Localized Containers are discussed in Chapter 10 of the **GoldSim User's Guide**.

There is no limit to the degree of hierarchy you can build in this way. For example, a Car system might contain an Engine and a Transmission system. The Engine system within the car might have 4 Cylinder components, and the Transmission system 6 Gearset components:



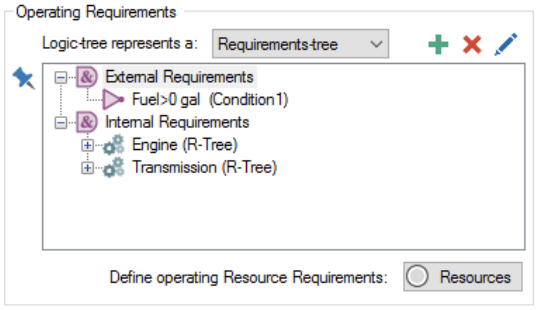
When you build a system with sub-components, the parent (the system) and the children (the sub-components) can interact in three important ways:

- If the parent stops operating, the children immediately stop operating.
- If you specify any internal requirements for the parent (in the Operating Requirements portion of the dialog), these can act to stop the parent from operating based on whether or not the children are operating.
- For Action elements, the parent can delegate processing of its actions to its children.

Read more: Specifying that Actions are to be Handled Internally (page 78).

Operating Requirements

The "Operating Requirements" portion of the reliability element dialog allows you to define logical relationships between elements using a logic tree:



GoldSim allows you to specify either a requirements-tree (the default) or a fault-tree. A requirements tree must evaluate to true in order for the element to operate, while a fault tree must evaluate to false in order to operate.

There are two types of operating requirements: external and internal. External requirements refer to the status of other elements in the model that are not sub-components of the given element. Internal requirements are used to specify the effects of failure modes on operability (by default, failure modes are added as internal requirements and hence cause the element to stop operating), and can also refer to the status of child elements (if the element is modeled as a system).

Read more: Defining Operating Requirements for Reliability Elements Using Logic Trees (page 64).

In addition to specifying operating requirements, you can also specify Resources that are required in order for the element to operate (via the **Resources** button).

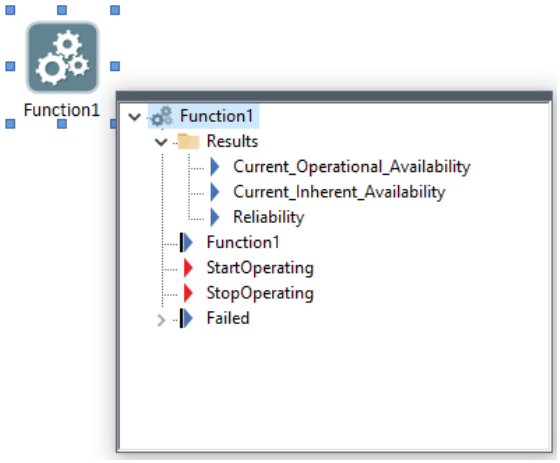
Read more: Modeling Resources in the Reliability Module (page 121).

The Common Outputs and Locally Available Properties of the Reliability Elements

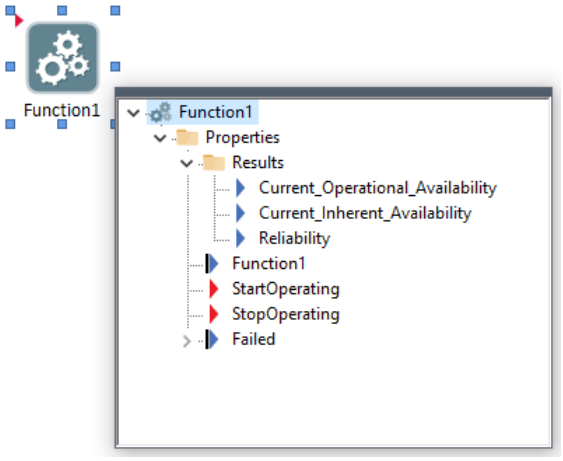
The two reliability elements (Functions and Actions) have a number of common outputs and locally available properties. These are discussed in detail in the following sections.

Common Reliability Element Outputs

The Function and Action component share seven outputs:



Note that if the element is being modeled as a system with child elements, these outputs are contained within a "Properties" folder when viewing the output port for the element:



The Function and Action element's primary output (the output that takes on the same name as the element) is an integer that indicates the Status of the component. At any given time, it takes on one of the following values:

Output Value	Component Status
0	All requirements are met, the component is not failed, it is turned on and operating.
1	A preventive maintenance (that makes the component inoperable) is underway.
2	Internal requirements (including failure modes) are not met.
3	External requirements are not met.
4	Element is not turned on.
5	Parent element is not operating.
6	An operating Resource requirement is not met.

If more than one status value applies at any given time, then the smallest value is output.

Read more: Example: Using the Reliability Element's Primary Output (page 146); Modeling Maintenance in the Reliability Module (page 114); Modeling Resources in the Reliability Module (page 121).

Because it is quite common to reference the status of a component, the Reliability Module provides some built-in constants to make it easier to read and recognize expressions that reference these values:

Constant	Value
RL_Operating	0
RL_PM	1
RL_IntReqFail	2
RL_ExtReqFail	3
RL_Off	4
RL_ParentNotOp	5
RL_NotOpResources	6

Hence, if you wanted to write an equation that checked whether or not the Function element "Computer" was operating, you could write either of the following conditional expressions:

Computer = 0; or

Computer = RL_Operating

Built-in constants can be entered directly, or can be inserted by selecting "Constants" from an input field's context menu (accessed by right-clicking in any input field).

Immediately below the Status output in the output port for a reliability element are found two discrete event signal outputs:

StartOperating: An event that is emitted when the element starts operating.

StopOperating: An event that is emitted when the element stops operating.

Read more: Understanding Discrete Events and Triggering (page 18).

Below these two outputs you will also find the following output:

Failed: This is a vector of ten items. It is based on an array label set named FailureModes (with ten items from 1 to 10) that is automatically provided by GoldSim. Each item in the vector corresponds to a failure mode defined for the element (GoldSim allows up to 10 failure modes). For example, Failed[1] corresponds to the failure mode #1. Each item is set to True if its corresponding failure mode has occurred (and it is reset to False if the mode is repaired).

By default, the Failed output is not set to save time history results (since it is an array). To save time history results for the Failed output, you must connect it to a Time History Result element or right-click on it in the output interface and select the Save Time History option.

Read more: Defining Failure Modes (page 92); Reliability Element Status and Failure Mode Histories and Statistics (page 141).

Finally, within the output port for a reliability element there are three additional outputs provided in a folder named Results:

Current_Operational_Availability: This is the fraction of time the element was operating over a specified time period immediately previous to the current time. The specified time period is approximately 2% of the simulation duration.

Current_Inherent_Availability: This is the fraction of time the element was operable over a specified time period immediately previous to the current time. A component that is turned off or has unmet external requirements, but has not failed would be considered operable. Hence, the Inherent Availability is always greater than or equal to the Operational Availability. The specified time period is approximately 2% of the simulation duration.

Reliability: This value is 1 if the element has not failed, and 0 if it has failed. Unlike the availability results, no averaging period is required. At any point in time, the element has either failed or has not failed.

Read more: Availability and Reliability Histories and Statistics (page 134).

Note: These two availability results represent current (or, conceptually, instantaneous) values. They are different from and should not be confused with the availability results that are presented on the Results tab of Reliability elements, which represent the availability over the entire duration of the simulation.

Read more: Availability and Reliability Results Summary (page 132).

Common Reliability Element Locally Available Properties

Locally available properties are special attributes of some elements in GoldSim. Locally available properties are similar to element outputs in that they have a data type (e.g., value, condition), order (e.g., scalar, vector) and dimensions (i.e., units). However, they do not appear as outputs of the element to which they belong. Rather, they are only visible in browsers (the main browser, or the Insert Link browser within their parent element).

Note: Locally available properties are only shown in the main browser if you choose to **Show element subitems** (accessed via the browser context menu by right-clicking in the browser).

Locally available properties derive their name from the fact that they may only be available, or they may take on different values (i.e., be over-ridden), in "local" parts of your model (e.g., within a particular element, or within a particular input field for an element).

Note: Locally available properties are discussed in detail in Chapter 10 of the **GoldSim User's Guide**.

The Function and Action components provide five outputs in the form of locally available properties:

Status. This is the current value of the main output of the reliability element.

OpTime. A value indicating the cumulative time that the element has been operating since the start of the simulation or its last Replacement event.

OpTimeSincePM. A value indicating the cumulative time that the element has been operating since the start of the simulation or its last Replacement event, or since the last time the component has completed a PM: Preventive Maintenance (failure mode) event.

TotalTime. A value indicating the time since the start of the simulation or the last replacement of the component.

Failed. This is a vector of ten items. Each item in the vector corresponds to a failure mode defined for the element (GoldSim allows up to 10 failure modes). For example, ~Failed[1] corresponds to the first failure mode. Each item is set to True if its corresponding failure mode has occurred (and it is reset to False if the mode is repaired).

Read more: Modeling Maintenance in the Reliability Module (page 114).

These five outputs can only be referenced within the reliability element itself (i.e., within a property dialog of the element, or if it is a system, within the system) by adding the ~ prefix. For example, if you wanted to reference the Status of the element inside a reliability element dialog (e.g., to define a failure mode), you would have to reference "~Status".

Note: The locally available properties Status and Failed are also outputs of the element. If you want to reference the Status *outside* of the element, you should use the element name (the Status is the element's primary output). If you want to reference this variable inside the element, you should use "~Status". Similarly, if you want to reference the Failed vector *outside* of the element, you should use "*elementID*.Failed". If you want to reference this variable inside the element, you should use "~Failed".

Note: The most common example of the use of a local variable can be seen by looking at an element's Operating Requirements. When you add a failure mode to an element, GoldSim automatically adds the failure mode as an Internal Requirement in the element's Requirements tree. It actually does this by inserting a "Not" condition: Not ~Failed[n], where n is the failure mode. ~Failed[n] is a local variable.

Read more: Failure Modes and Internal Requirements (page 95).

If a component is defined as a system, its locally available properties are available inside the system (but not within the dialogs of child reliability elements inside the system). For example, if you created an Expression element inside a reliability element defined as a system, and defined it as "~Status", this would return the Status *of the parent component*. However, if you referenced "~Status" inside the dialog of a child reliability element inside the system, it would return the Status of the child (since the child's local property would override the parent's local property).

Read more: Modeling a Reliability Element as a System with Child Elements (page 58).

In addition to the use of the **~Failed** variable in an element's Internal Requirements, other examples of where it could be useful to reference locally available properties within a reliability element include the following:

1. In the case of non-fatal failure modes (in which you have manually removed the Not ~Fail[n] condition for that mode from the Internal Requirements), you may want to specify that the failure, while not fatal, reduces the capacity or performance of some internal component of the system by referencing **Failed[n]** in an ancillary calculation (e.g., throughput).
2. You may want to trigger a preventive maintenance "failure mode" based on the value of **OpTimeSincePM** or **TotalTime**.
3. You may want to accelerate or decelerate a failure mode based on the **Status** of the element or based on the value of **OpTime** or **TotalTime**.

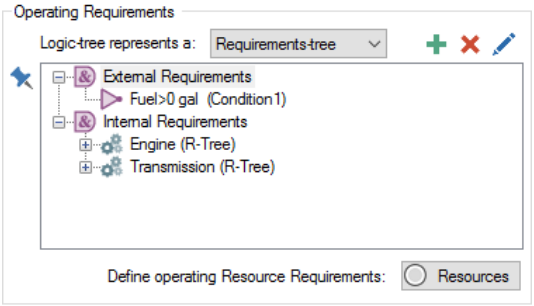
Read more: Modeling Coupled and Non-Fatal Failure Modes (page 105); Simulating Preventive Maintenance as a Failure Mode (page 114); Modeling Acceleration for Failure Mode Control Variables (page 112).

Three other locally available properties (FMCV, FM_Failed, and FM_TimeToFail) are available from within input fields for certain failure mode input parameters. The former is discussed elsewhere while describing FMCVs. The latter two are discussed with regard to simulating preventive maintenance as a failure mode.

Read more: Referring to FMCVs When Defining Failure Mode Parameters (page 114); Simulating Preventive Maintenance as a Failure Mode (page 114).

Defining Operating Requirements for Reliability Elements Using Logic Trees

The "Operating Requirements" portion of the reliability element dialog allows you to define logical relationships between elements using a logic tree:



Logic trees are defined using a number of different types of nodes (e.g., AND gates, OR gates).

In order for a reliability element to be operable, all of its defined requirements must be met. By default, no external requirements are defined, and all of the element's failure modes are added as internal requirements.

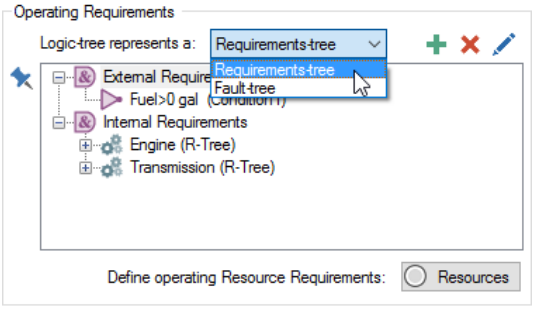
The sections below describe the details of how you can create logic trees for a reliability element.

Read more: Example: Modeling Dependencies on Other Reliability Components (page 148); Example: Working with Internal and External Requirements (page 152).

The Two Types of Logic Trees

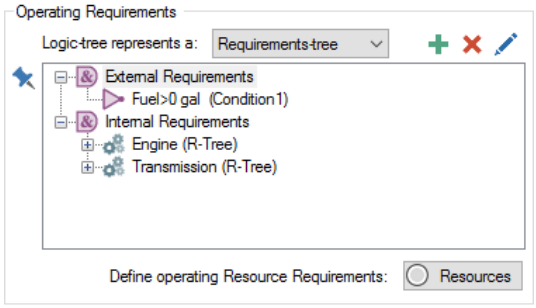
There are two types of logic trees that can be defined for a reliability element: a requirements-tree (the default) or a fault-tree. A requirements tree must evaluate to true in order for the element to operate, while a fault tree must evaluate to false in order to operate.

The type of tree is selected using a drop down list in the requirements dialog:

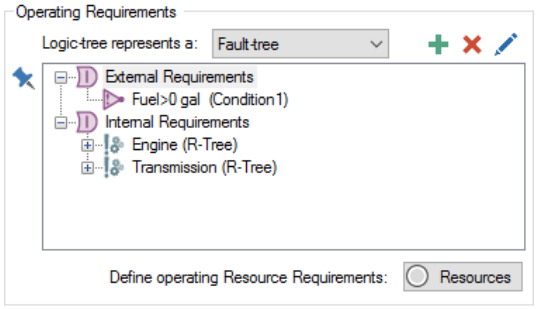


In addition to supporting both types of logic-trees, GoldSim allows you to automatically convert from one logic-tree type to another. For example, if you have defined a requirements-tree, changing the drop down selection to a fault-tree will automatically convert the nodes in the requirements-tree into an equivalent fault-tree and vice versa.

Here is that same Operating Requirements, presented first as a Requirements tree:



and then as a fault tree:



Note that And gates have changed to Or gates, and the not (!) symbol appears before the other items. Recall that a requirements tree must evaluate to true in order for the element to operate, while a fault tree must evaluate to false in order to operate.

Read more: Understanding Logic Tree Nodes (page 68).

External and Internal Requirements

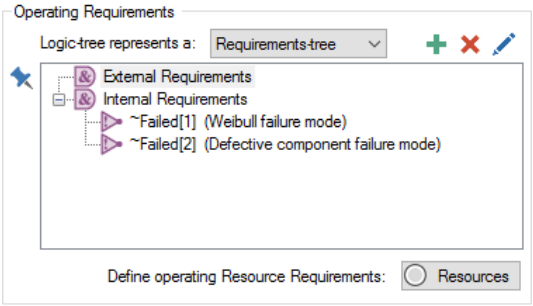
There are two types of operating requirements: external and internal. External requirements refer to the status of other elements in the model that are not sub-components of the given element.

For example, assume that we define a computer as a system, and place inside the system several sub-components (e.g., motherboard, CPU, power supply, DVD drive, monitor). Specifying that a motherboard can operate only if the power supply is operating is an external requirement with respect to the motherboard (since the power supply is external to the motherboard).

Internal requirements reference things internal to the element. There are two types of things that are considered to be internal to a Reliability element:

1. Failure modes for the element. By default, any failure modes for the element are automatically added as internal requirements. In particular, when you add a failure mode to an element, GoldSim automatically inserts a "Not" condition in the internal requirements portion of the requirements tree: Not ~Failed[n], where n is the failure mode.

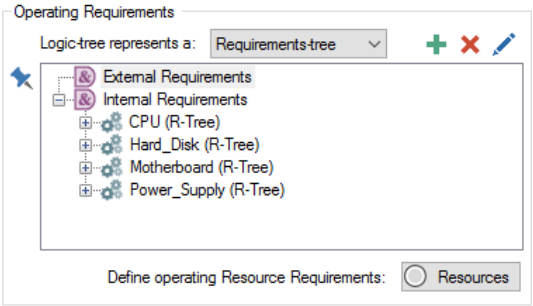
   In the example below, there are two failure modes for the element, and these are added automatically to the requirements tree:



Note: Adding a failure mode as an internal requirement implies that it is assumed to be fatal to the component (i.e., if the failure mode occurs, the component itself fails and is no longer operative). However, if desired, you can specify that a failure mode is non-fatal (by manually removing it from the requirements tree) in order to model more complex failure mode behavior.

Read more: Failure Modes and Internal Requirements (page 95); Condition and Not Nodes (page 71); Modeling Coupled and Non-Fatal Failure Modes (page 105).

2. Child elements of a component that is being modeled as a system. You can manually refer to the operating status of child elements. For example, specifying that a computer can operate only if the power supply is operating is an internal requirement with respect to the computer (since the power supply is internal to the computer):



Read more: Modeling a Reliability Element as a System with Child Elements (page 58).

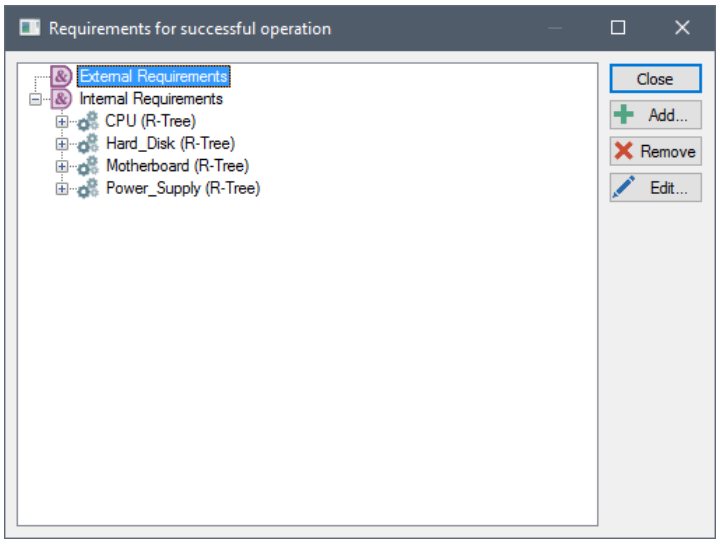
Expanding the View of the Operating Requirements

Pushpin button

You can choose to view and edit the requirements tree in two different ways.

The default view is the one that appears when the reliability element's dialog is first opened. In this case, the operating requirements are incorporated directly into the dialog itself.

If you have a very large logic tree with a number of hierarchical levels, it might be useful to view the tree in an expanded sub-window. You can do this by pressing the "pushpin" button in the Operating Requirements portion of the dialog. When you do so, GoldSim will display the tree in a separate (and resizable) window:



Clicking the **Close** button in the upper-right corner of the dialog exits the expanded viewing window.

Adding, Removing and Editing Nodes in the Logic-Tree

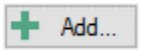
When viewing the operating requirements, there are three buttons which allow you to edit the logic tree.

The **Add** button allows you to add nodes (Gate nodes or variable nodes) to the logic tree.

Read more: Understanding Logic Tree Nodes (page 68).

It appears as:

 in the standard reliability dialog, and

 in the expanded sub-window.

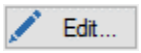
The **Remove** button allows you to delete gates and conditions from the logic tree. It appears as:

 in the standard reliability dialog, and

 in the expanded sub-window.

The **Edit** button allows you to edit certain types of nodes (variable nodes) in the logic tree. It appears as:

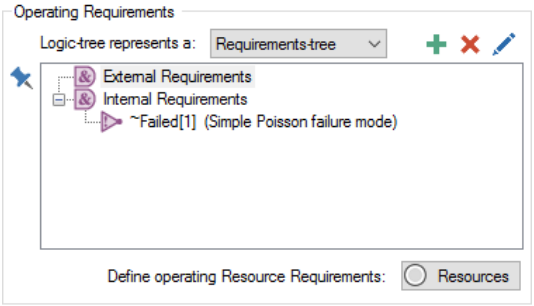
 in the standard reliability dialog, and

 in the expanded sub-window.

Selecting a variable node type and clicking the Edit button (or double-clicking the node) displays the same dialog that appeared when the node was added.

Understanding Logic Tree Nodes

When a reliability element is initially created, the Operating Requirements logic tree will have two AND-Gates (one for External Requirements and one for Internal Requirements). The External Requirements AND-Gate will be empty, and the Internal Requirements AND-Gate will include a Not node referencing the default failure mode for the element:



The External Requirements AND-Gate can be used to reference conditions/requirements outside the element, while the Internal Requirements AND-Gate can be used can be used to reference child elements and conditions (such as the status of a failure mode).

Read more: External and Internal Requirements (page 66).

A logic tree consists of two types of nodes: gate nodes and variable nodes.

Gate nodes are used to construct the logical framework of the tree. Gate nodes require "child nodes" that are either other gate nodes or variable nodes.

There are three types of gate nodes:

AND-Gate. An AND-Gate is true when all of its child nodes are true. The child nodes can be variable nodes, or other gate nodes.

OR-Gate. An OR-Gate is true when at least one of its child nodes is true. The child nodes can be variable nodes, or other gate nodes.

N-Vote Gate. An N-Vote Gate is true when N (which is user-specified) or more of its inputs are true**.** The child nodes can be variable nodes, or other gate nodes.

There are four types of variable nodes, each of which requires an input:

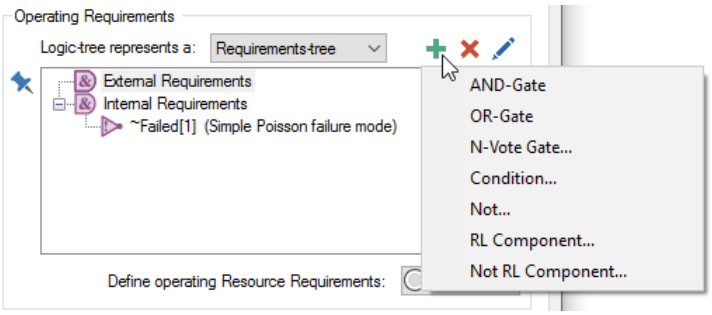
Condition. A Condition node is true when the specified condition evaluates to true. The input must be a Condition output or a conditional expression. The Condition node is often used to reference the status of the element or a failure mode.

Not. A Not node is true when the specified condition evaluates to false. The input must be a Condition output or a conditional expression. The Not node is most commonly used to reference the status of the element's failure modes.

RL Component. An RL Component node is true when the referenced reliability element is operating (when referencing reliability element as an External Requirement) or capable of operating (when referencing a child reliability element as an Internal Requirement).

Not RL Component. A Not RL Component node is true when the referenced reliability element is not operating (when referencing a reliability element as an External Requirement) or not capable of operating (when referencing a child reliability element as an Internal Requirement).

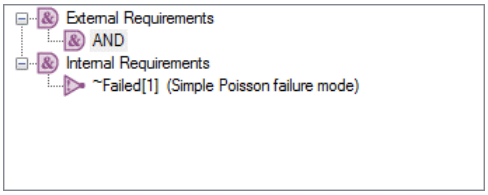
Clicking the **Add** button displays a menu which allows you to select the desired node type:



When adding a variable node, a dialog will be displayed for defining the input to the node.

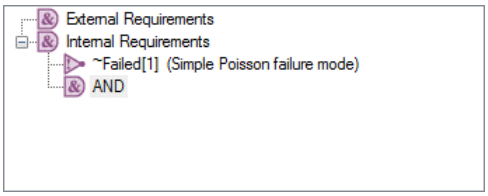
Where the new node will appear is determined by the type of node that is selected prior to pressing the **Add** button:

- If a gate node is selected prior to pressing the **Add** button, the new node will appear as child node to the selected gate node:



In this example, the External Requirement AND-Gate was selected when an AND-Gate was added. The AND-Gate is therefore added as a child node to the External Requirements node.

- If a variable node is selected prior to pressing the **Add** button, the new node will appear as child node to the selected node's parent:



In this example, a Not Condition node (the default Not node referencing the default failure mode) was selected when an AND-Gate was added. The AND-Gate is therefore added to the parent of the Condition node.

GoldSim allows you to automatically convert from one logic-tree type to another. The automatic conversion is carried out as follows:

Original Node…	Is converted to this
Condition	Not
Not	Condition
AND-Gate	OR-Gate
OR-Gate	AND-Gate
N-Vote Gate (N or more must be true)	N-Vote Gate (total child nodes – N + 1 or more must be true)
RL Component	Not RL Component
Not RL Component	RL Component

Adding and using AND and OR Gates is straightforward. Use of the other nodes is described in more detail in the sections below.

N-Vote Gates

An N-Vote Gate is true when N or more of its inputs are true. The inputs can be conditions, or other gates. With an N-Vote gate, you need to specify the number of inputs that must be true in order for the gate to be true. The following dialog appears when the N-Vote gate option is selected:

The number of required votes can be edited by typing directly into the text box, or by using the arrows to the right of the text box to increment the number of votes up or down. The number of votes must be an integer, so non-integer numbers are automatically truncated. Once the node has been placed, it will display the number of votes required, as shown below:

Condition and Not Nodes

The input for Condition and Not nodes must be a Condition output or a conditional expression. A Condition node is true when the specified condition evaluates to true. A Not node is true when the specified condition evaluates to false.

When a Condition or Not node is added, a small dialog appears:

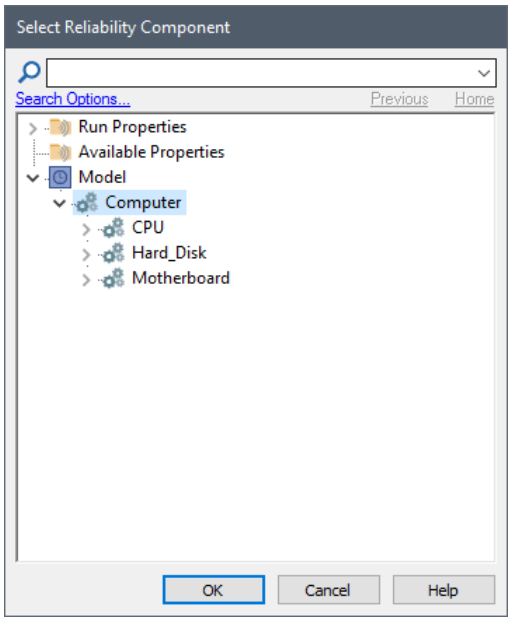
This dialog allows you to enter a conditional expression (an expression which evaluates to true or false), or create a link to a condition-type output of another GoldSim element.

RL Component Nodes

An RL Component node is true when the referenced reliability element is operating (when referencing reliability element as an External Requirement) or capable of operating (when referencing a child reliability element as an Internal Requirement).

A Not RL Component node is true when the referenced reliability element is not operating (when referencing reliability element as an External Requirement) or not capable of operating (when referencing a child reliability element as an Internal Requirement).

When either one of these node types is added, a browser listing all of the reliability elements in the model will appear:



Selecting the appropriate reliability element and pressing **OK** will create the new RL node.

Note: When referencing a child reliability element as an Internal Requirement using an RL Component node, the node will evaluate to true when the reference child element is either operating, or capable of operating, but inoperable because the parent is inoperable or undergoing a preventive maintenance. Likewise, when referencing a child reliability element as an Internal Requirement using a Not RL Component node, the node will evaluate to true when the reference child element is not capable of operating.

Note: Child reliability elements can not be referenced by a parent if they are contained inside a localized Container (or another reliability element being modeled as a system) within the parent.

Editing Other Element's Logic Trees from a Dependent Element

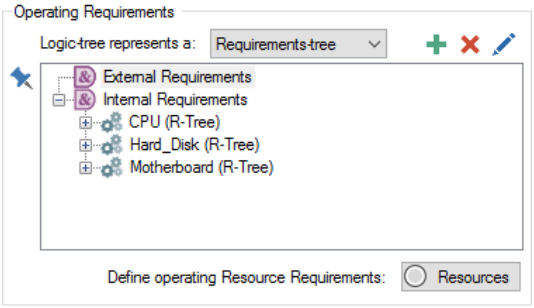
The logic tree dialog allows you to edit the logic trees of other reliability elements that are linked using RL Component nodes to the logic tree being edited.

To illustrate this, consider a simple example in which a Computer is modeled as a system consisting of a motherboard, hard disk and CPU, in which all three must be operating in order for the Computer to operate. Furthermore, the CPU is modeled as a system consisting of an onboard fan, which must operate in order for the CPU to operate.

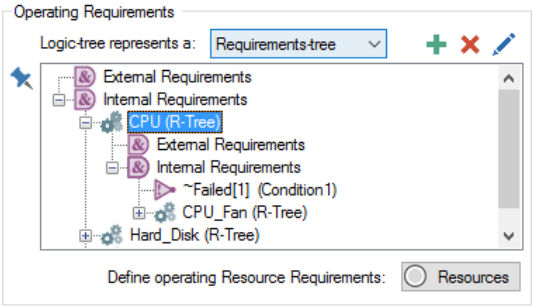
To implement these logical relationships, we could construct a logic tree for the Computer element, and then open the CPU element and construct a logic tree that had an RL component node linked to the CPU fan.

But we don't actually have to leave the Computer element's dialog to accomplish this. That is, instead of building the two trees independently, GoldSim allows you to construct the entire tree from the Computer element's dialog.

Let's look at the Computer element's dialog after we have added RL component nodes for the CPU, Hard Disk and Motherboard:

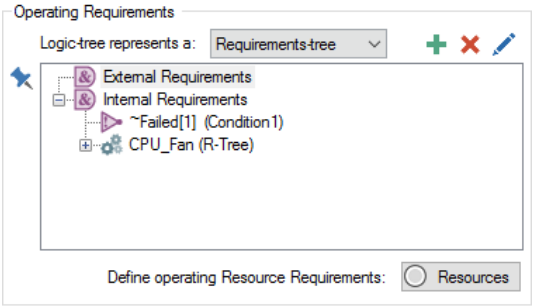


Clicking on the plus sign next to *any* RL component will display the linked RL element's requirements tree, and we can add the CPU's dependency on the CPU fan by adding an RL Component node to the CPU's logic tree:



Notice how the CPU fan RL Component node also has a plus sign next to it. You can continue to browse and edit the logic trees of other elements down to an arbitrary level of complexity.

Of course, if we subsequently opened the CPU element, its portion of the logic tree would reflect the changes made within the Computer element:



Specifying Operating Resource Requirements

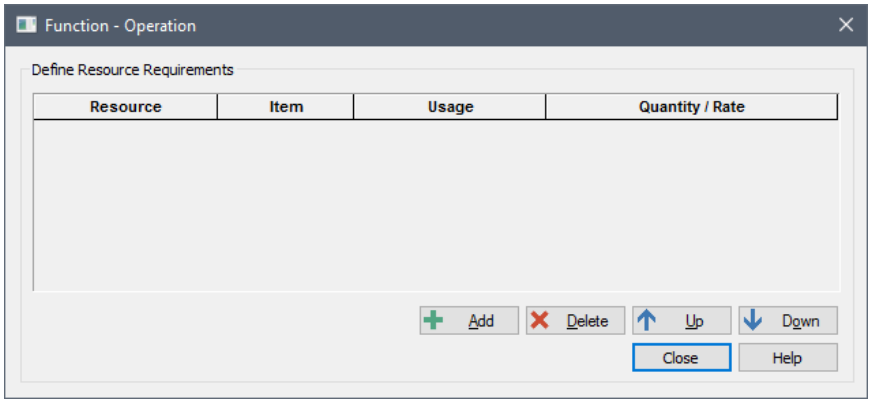
A ***Resource*** is something that has a limited supply (e.g., spare parts, fuel, skilled personnel, money) and is required in order for elements of the modeled system to carry out certain actions.

Read more: Modeling Resources in the Reliability Module (page 121).

For Function and Action elements, you can specify that certain Resources must be available in order for the component to operate. In particular, in order for the component to operate, it may need to utilize Resources at a specified rate (e.g., 10 gal/hr of Fuel). If the Resources can not be provided at the specified rate, the component will stop operating until they are available.

To define an operating Resource Requirement for a Function or an Action, press the **Resources** button at the bottom of the Operating Requirements section of the

dialog (labeled “Define operating Resource Requirements”). The following dialog will be displayed:



Note: In order for this dialog to appear, you must have previously defined at least one Resource in the model.

You can add a Resource Requirement by pressing the **Add** button.

An operating Resource Requirement interacts with the specified Resource Stores when the element is operating, and can only have two types of interactions (specified in the **Usage** column):

- **Spend At Rate:** A specified continuous spend rate of the Resource is required in order for the element to continue to operate. If the requested Resource is not available in sufficient quantity to maintain the requested spend rate, the element stops operating. If it becomes available again, the element starts operating again (assuming all other requirements are met).
- **Deposit At Rate:** While the element is operating, the specified deposit rate is added to the Store.

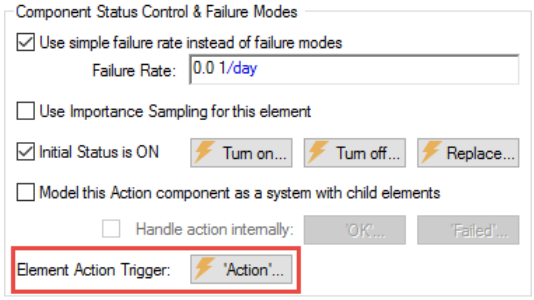
Inputs, Outputs and Features Specific to the Action Element

The Action element has a number of inputs and features that it does not share with the Function element. These are discussed in detail in the following sections.

Triggering the Action Element

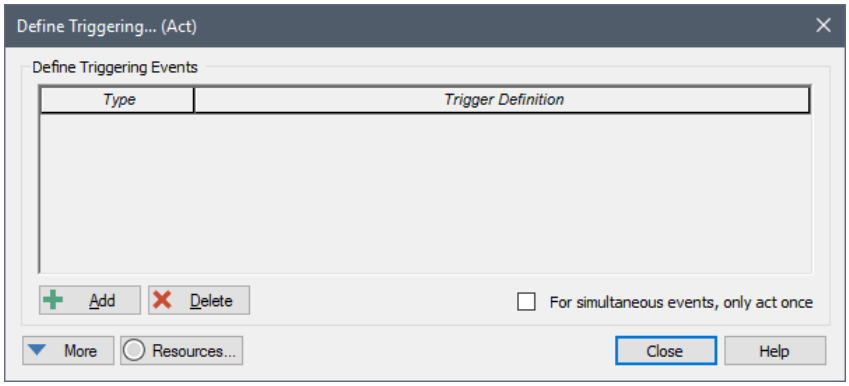
Action elements represent activities or processes that are carried out discretely (as opposed to continuously) such as when a door latches closed, a switch opens or closes, an engine starts, or a message is delivered. The Action component waits for a triggering input to tell it to carry out its action. If its action succeeds, the element emits an ‘ActionOK’ event, and if it fails the element emits an ‘ActionFailed’ event.

Hence, a key input for the Action element is the **Action Trigger**:



This trigger is used to specify the condition or event which will "trigger" the Action element to carry out its action.

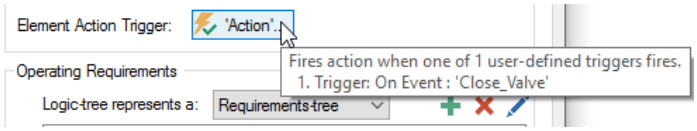
Clicking the **Action…** button brings up the standard GoldSim triggering dialog:



This dialog allows you to specify conditions that will result in the element's Action being triggered.

Read more: Defining Triggers (page 20).

Note that once a trigger has been specified, a green check appears in the bottom left corner of the **Action…** button, and mousing over the button will display any triggers which have been specified:



Note: If the Action element is turned off when it receives an Action trigger, the trigger is ignored.

Specifying Resource Requirements for Triggering an Action

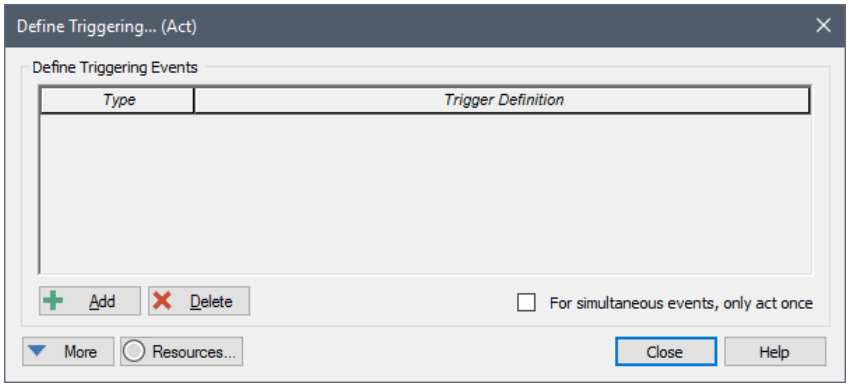
Within the dialog for triggering an Action, you can also specify Resource Requirements necessary for the Action to be triggered.

A ***Resource*** is something that has a limited supply (e.g., spare parts, fuel, skilled personnel, money) and is required in order for elements of the modeled system to carry out certain actions.

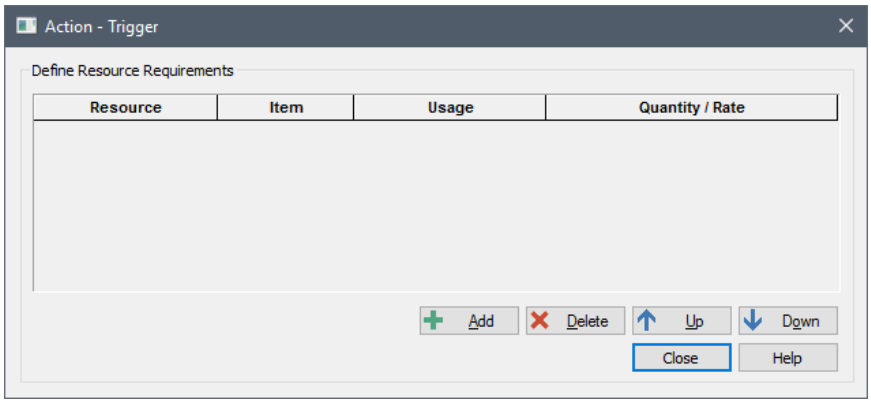
Read more: Modeling Resources in the Reliability Module (page 121).

When triggering an Action element, you can specify that certain Resources must be available in order for the component to be triggered. If the Resources are not available when the element is triggered, the Action trigger will fail.

To define Resource Requirement for triggering an Action, press the **Resources** button in the Trigger dialog:



The following dialog will be displayed:



Note: In order for this dialog to appear, you must have previously defined at least one Resource in the model.

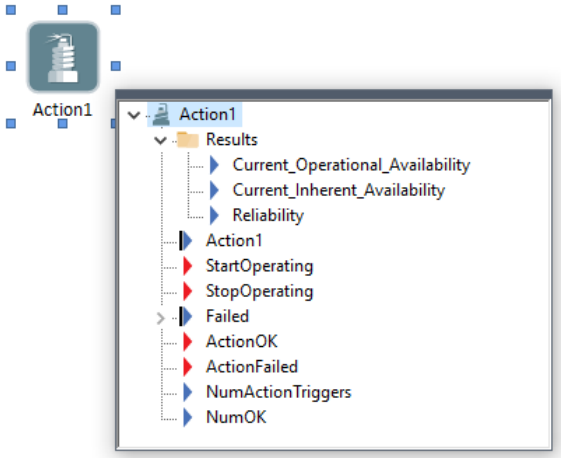
You can add a Resource Requirement by pressing the **Add** button.

An Action trigger interacts with the specified Resource Stores when it is triggered, and can only have three types of interactions (specified in the **Usage** column):

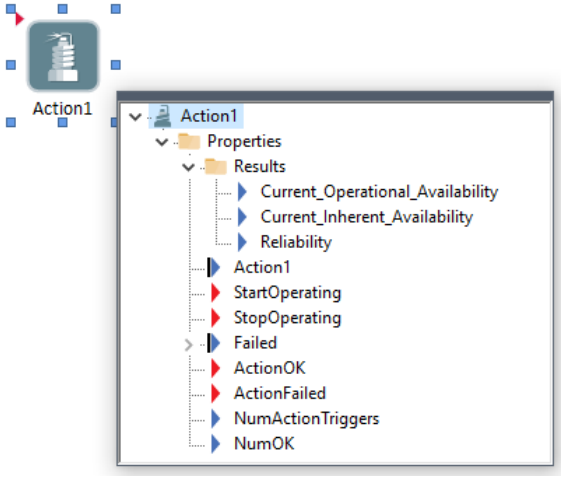
- **Spend (discrete):** A discrete quantity of the Resource is required in order to complete the Action trigger. If the requested Resource quantity is not available, the trigger signal is "held", and it waits for the Resource to become available.
- **Borrow (discrete):** A discrete quantity of the Resource is required in order to complete the Action trigger. If the requested Resource quantity is not available, the trigger signal is "held", and it waits for the Resource to become available. If the Resource is available, the Action is triggered and the borrowed quantity is returned to the Resource Store. (If the Action has no specified Delay, the Resource is effectively returned immediately.)
- **Deposit (discrete):** A discrete quantity of the Resource is created and deposited with the Store when the Action is triggered.

Outputs Available Only for Action Elements

The Action element has 11 outputs:



Note that if the element is being modeled as a system with child elements, these outputs are contained within a "Properties" folder when viewing the output port for the element:



The first seven outputs are shared with the Function element.

Read more: Common Reliability Element Outputs (page 60).

The other four outputs are unique to the Action element:

ActionOK. An event that is output whenever the element is triggered and successfully performs its action.

ActionFailed. An event that is output whenever the element is triggered and fails to successfully perform its action. The event is output whether the action failed due to the element not being operating or if it failed because the element was unreliable.

NumActionTriggers. An integer representing the cumulative number of times the element has been triggered.

NumOK. An integer representing the cumulative number of times the element has successfully carried out its action after being triggered.

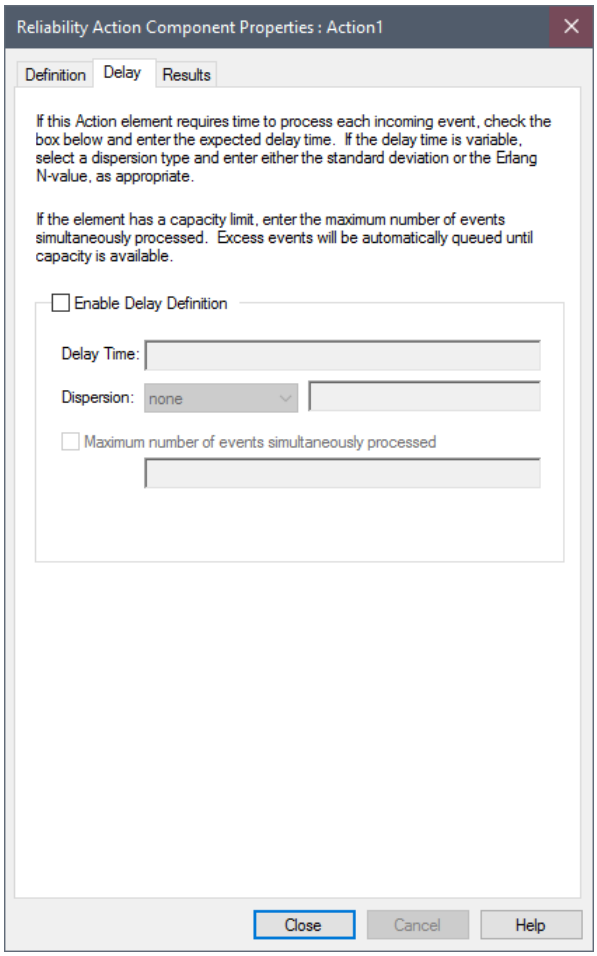
In addition, under some circumstances (if the Action is being modeled as a system, and actions are handled internally), a locally available property called ActionEvent is available for Trigger inputs inside the Action element.

Note: Locally available properties are discussed in detail in Chapter 10 of the **GoldSim User's Guide**.

Read more: Handling Actions Internally (page 125).

The Delay Tab of the Action Element

In some cases, it may take time for an Action element to complete its action. GoldSim provides features that allow you to model this, and these are accessible via the **Delay** tab available in the Action element's property dialog:

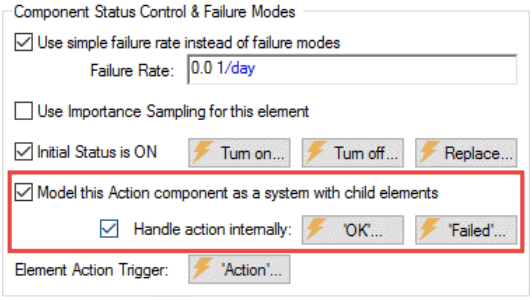


By default, an Action is instantaneous. To enable delay features in an Action element, you must check the **Enable Delay Definition** box.

Read more: Adding Delays to Action Elements (page 123).

Specifying that Actions are to be Handled Internally

Within the Action element dialog, there is an option to **Handle action internally**:



This option is only available if the Action is being modeled as a system.

Read more: Modeling a Reliability Element as a System with Child Elements (page 58).

Even if the Action is being modeled as a system, the checkbox is defaulted off.

This checkbox controls how Actions that are defined as systems handle action triggers. If the box is checked, the Action triggers are automatically routed to internal Action components and the OK and Failed triggers become available (for handling responses from internal elements).

Read more: Handling Actions Internally (page 125).

Chapter 4: Running a Reliability Simulation

I have not failed. I've just found 10,000 ways that don't work.

Thomas Alva Edison

Chapter Overview

This chapter describes how to set up and run a reliability model in GoldSim. For the most part, the information presented here is discussed in detail in Chapter 7 of the **GoldSim User's Guide**. This chapter summarizes that information, and adds some comments that are specific to running reliability models.

In this Chapter

This chapter discusses:

- Dynamic Reliability Modeling
- Using Monte Carlo Simulation in Your Reliability Model
- Static Reliability Modeling

Dynamic Reliability Modeling

Dynamic simulation allows you to develop a representation of the system whose reliability is to be determined, and then observe that system's performance over a specified period of time.

The primary advantages of dynamic simulation are:

- The system can evolve into any feasible state and its properties can change suddenly or gradually as the simulation progresses; and
- The system can be affected by random (stochastic) processes, which may be either internal (e.g., failure modes) or external.

In order to run a dynamic simulation, you must specify the duration of the simulation (e.g., 1 month, 1 year) and the length of the timesteps that you will use (i.e., the degree to which time will be discretized).

The decision on the simulation duration should be driven by the system variability you wish to capture with your model, since the duration is the amount of simulated time over which the system's components will be allowed to interact and evolve as the real system would. Typically you would use something such as the system's serviceable life as the simulation duration.

Note, however, that depending on the system being simulated, there are several ways to approach the duration. If you have a system that has a fixed service life (and the failure mechanisms have similar time scales to the service life), a dynamic *Monte Carlo simulation in which the duration is the service life of the system* would be the appropriate way to capture the behavior of the system. However, if you have a system that operates for a very long time period with repeated failures and repairs (such that it effectively reaches a "steady-state" condition), you may want to *run a single realization of the system with a very long duration*. Over a long duration, the variability in the failures and repairs will be adequately represented without running Monte Carlo simulation.

The specified number of timesteps is the minimum number of times that GoldSim will recalculate and update all of the elements in the model. The number of timesteps required to accurately model a system depends to a large extent on how you've built your model.

Note that GoldSim will interrupt the simulation and force it to update when a repair or failure occurs, even if that failure or repair does not occur on a fixed timestep.

Read more: How Failures and Repairs are Represented in Time (page 83).

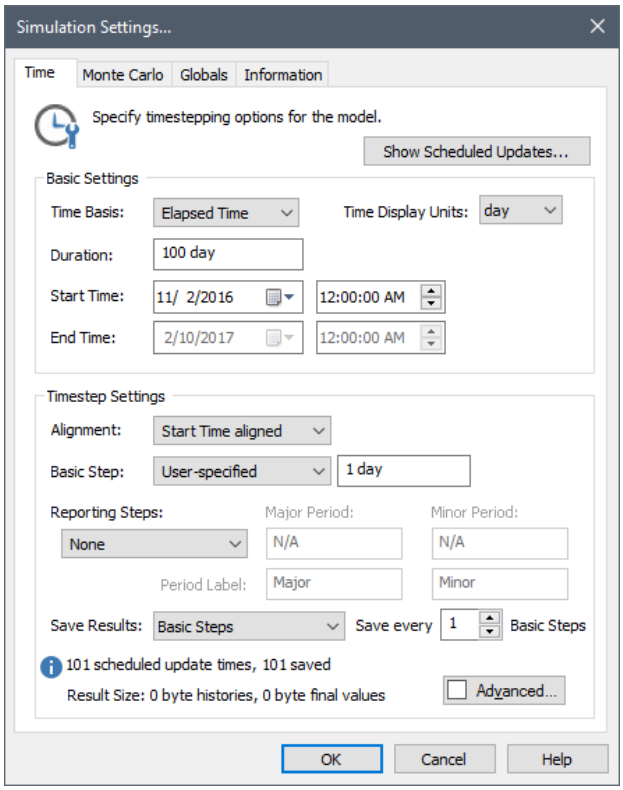
Nevertheless, some models may require a significant number of timesteps in order to generate valid results. For example, if your reliability elements have failure and repair mode parameters that vary dynamically, have acceleration factors which vary dynamically, utilize user-defined failure modes, or have logic trees linked to standard GoldSim elements such as Reservoirs or Pools, you need to ensure that the number of timesteps is sufficiently large that these dynamic changes are represented accurately.

Note: Details of GoldSim's dynamic timestepping algorithm, including a discussion of selecting the proper timestep for a model, are presented in Appendix F of the **GoldSim User's Guide**.

Setting Up a Dynamic Reliability Simulation

The basic time options in GoldSim are defined in the **Time** tab of the Simulation Settings dialog. The Simulation Settings dialog is accessed by pressing **F2** or by selecting **Run | Simulation Settings…** from the main menu.

The **Time** tab of the Simulation Settings dialog is shown below:



The **Duration** of the simulation can be specified as a time period (e.g., 5 years), or in terms of a start and end date in the Basic Time Settings section of the dialog. The length of the timestep (**Basic Step**) is specified in the Timestep Settings portion of the dialog.

GoldSim also provides a variety of advanced timestepping options, which are accessed via the **Advanced…** button in the **Time** tab.

Note: The timestepping options in GoldSim are discussed in detail in Chapter 7 of the **GoldSim User's Guide**.

How Failures and Repairs are Represented in Time

When a repair or failure is generated in GoldSim by a Function or Action element, the failure or repair may not fall exactly on a "scheduled" timestep (i.e., a timestep that was defined in the **Time** tab of the Simulation Settings dialog). That is, the failure or repair event may actually occur between scheduled timesteps.

These trigger an "unscheduled update" of the model. Unscheduled updates are timesteps that are dynamically inserted by GoldSim during the simulation in order to more accurately simulate the system. That is, they are not specified directly prior to running the model. GoldSim inserts them automatically (and, generally, without you needing to be aware of it).

For example, if you had specified a one day timestep, and failure occurs at 33.65 days (i.e., between the scheduled one-day updates), GoldSim would insert an unscheduled update at 33.65 days.

Note: The manner in which GoldSim inserts timesteps in response to events is discussed in detail in Chapter 7 of the **GoldSim User's Guide**.

A key and important difference between scheduled updates and unscheduled updates is that scheduled updates are included in time history plots and tables (unless you choose to exclude them). Unscheduled updates, however, do not appear in time history plots and tables. That is, although these timesteps may affect the results (e.g., by making them more accurate at the scheduled timesteps), unscheduled updates of the model are not saved and plotted. Only the scheduled updates are actually saved and plotted.

Note: In some cases, it may be of interest to see the values of selected outputs that were computed at unscheduled updates. To facilitate this, Time History Result elements provide an option to do so.

Using Monte Carlo Simulation in Your Reliability Model

Once a model of a system has been constructed, you can simulate the system to predict how it will perform through time. By definition, however, the performance of any system involving reliability elements is stochastic (i.e., inherently variable), since failure is generally described stochastically. That is, we can't say exactly when a component will fail; we can only describe the failure (and repair) process statistically. For example, if we had 100 identical computers, their failure (and repair) histories would not be identical; they would display a distribution of behaviors.

In addition to this inherent variability, we might also be uncertain about some of the input parameters controlling the model. For example, if we had not carried out actual tests on the components, the parameters describing their failure modes would be uncertain, and we could enter these as probability distributions in order to capture this uncertainty.

Variability and uncertainty are represented in GoldSim using Monte Carlo simulation. Monte Carlo simulation consists of calculating a large number of "realizations" (potential futures). Each realization simulates the same system with the same initial conditions, but with different sampled stochastic values, both at the beginning of the simulation and as the system evolves through time. This results in a large number of separate and independent results, each of which is considered equally likely. These realizations can then be combined to provide statistical information on possible outcomes.

Note: If studying a system that is effectively at steady-state, it can be appropriate to run a single arbitrarily long simulation (e.g., 1000 years), as this can capture the variability in the failures and repairs. However, for a system that is aging, the actual life span of the system should be simulated using Monte Carlo simulation.

The number of realizations that are required in order to accurately capture the behavior of a system is a complex issue that can be influenced by the computational requirements of running a realization, and the frequency of the behaviors you wish to capture.

A rule of thumb for determining the number of realizations is that the number of realizations should be large enough that at least 10 simulations will have an occurrence of the most infrequent behavior you want to capture. For example, if you wanted to observe two consequences, one which occurred once in every 10 realizations, and another that occurred once in every 500 realizations, an adequate number of realizations for the simulation would be 5000.

Another factor that needs to be considered when deciding on the number of realizations is the impact that the number of realizations can have on the results and statistics available at the end of a dynamic simulation.

Read more: Availability and Reliability Results Summary (page 132).

In order to generate confidence bounds on the Inherent Availability, Operational Availability and Reliability metrics, at least six realizations must be run. However, this is a bare minimum, and if you are making use of these confidence bounds we strongly suggest running at least 100 realizations.

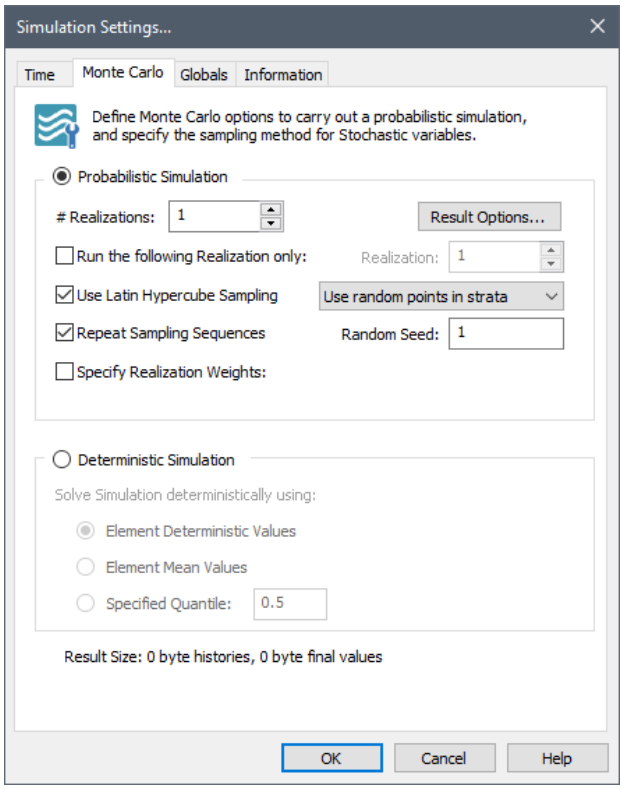
Furthermore, in order for the Failure Times results analysis to be available in result mode, the number of realizations must be large enough so that the sum of the number of failures over all realizations is greater than 5. Similarly, for the Repair Times analysis to be available, the sum of the number of repairs over all realizations must be greater than 5. As with the confidence bounds, the more failures and repairs that can be simulated, the less likely it is that the results represent an anomalous situation.

Read more: Failure Times Statistics (page 136).

Setting the Monte Carlo Options for a Reliability Model

Monte Carlo options in GoldSim are defined in the **Monte Carlo** tab of the Simulation Settings dialog. The Simulation Settings dialog is accessed by pressing **F2** or by selecting **Run | Simulation Settings…** from the main menu.

The **Monte Carlo** tab of the Simulation Settings dialog is shown below:



To change the number of realizations, simply edit the number in the **# Realizations** field.

Note: The Monte Carlo options in GoldSim are discussed in detail in Chapter 7 of the **GoldSim User's Guide**.

Note: When Latin Hypercube Sampling (LHS) is selected in this dialog, it is also applied to failure modes (as well as other random variables in GoldSim). However, in this case, LHS is only applied to the first occurrence of the failure. LHS is not applied to subsequent occurrences within the same realization.

Note: If you choose to run a Deterministic simulation, the reliability elements still retain their stochastic behavior. However, other GoldSim elements (e.g., Stochastic elements, Timed Event elements) in the model behave deterministically.

Static Reliability Modeling

In some cases, you may want to run GoldSim statically (i.e., without timestepping). A static simulation consists of a single calculation of the system in its steady-state condition. As such, it is equivalent to a conventional fault tree analysis (using Monte Carlo simulation instead of cut sets).

Static simulation cannot represent any true system dynamics, but is significantly faster than dynamic simulation (as GoldSim only needs to calculate element

values once as opposed to the hundreds or thousands of updates in a dynamic simulation). Hence, a properly constructed static model can provide useful information on the long-term availability of systems that do not have complex dynamic behavior.

When running a static reliability simulation, the following logic is used by GoldSim:

1. The system is assumed to have operated for an indefinite period of time and reached a steady-state with regard to failures and repairs.
2. To determine each reliability element's availability, GoldSim computes the steady state probability of being operable for each failure mode as follows:

   MTTF / (MTTF + MTTR)

   where MTTF is the Mean Time to Failure, and MTTR is the Mean Time to Repair.
3. For each realization, each failure mode in each reliability element randomly realizes its state (operating or failed). If a reliability element has a failure mode that has failed, that reliability element is considered failed for that static reliability simulation.
4. Based on the status of each reliability element, the system is considered to either be operating or failed for a particular realization. By specifying multiple Monte Carlo realizations, the mean availability can be computed.

Note: After calculating whether it has failed due to a failure mode, the reliability element then checks that its Operating Requirements are met. Within the requirements-tree, any nodes that reference other GoldSim elements use their time = 0 values.

Static reliability simulations have several important limitations:

- The automatic repair feature for all failure modes must be turned on in order to run a static reliability simulation (otherwise a fatal error will be displayed when you try to run the model).

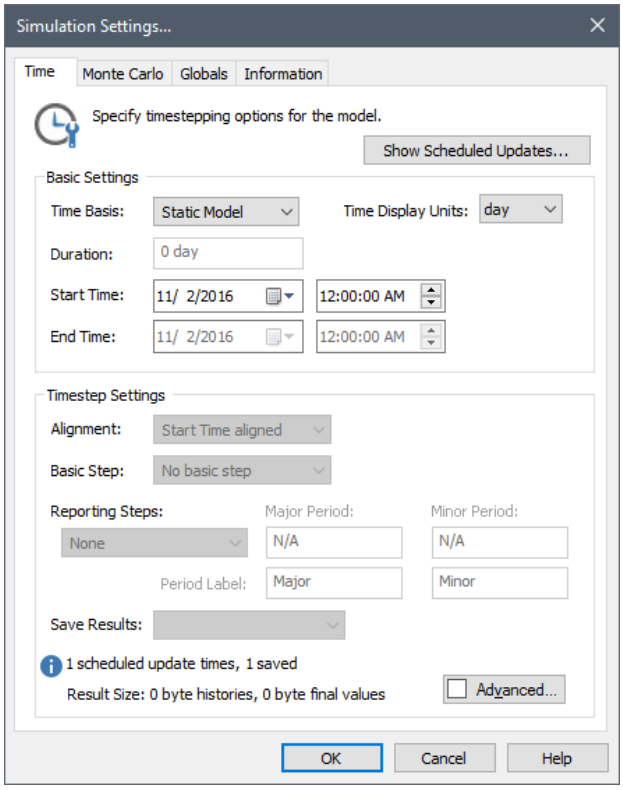
  Read more: Modeling the Repair of Failure Modes (page 101).
- Only failure modes that are FMCV-based (and do not have a user-defined base variable) are allowed (otherwise a fatal error will be displayed when you try to run the model).

  Read more: Failure Mode Control Variables (page 108).
- Static simulation only provides mean availability metrics. (It also provides reliability, but for a steady-state system, this is by definition always zero, unless there are no failure modes at all). No further results are available from a static simulation.
- Static models obviously do not allow the system to evolve in a natural way, and as a result, do not model system variability as accurately as a dynamic model.

Setting Up a Static Reliability Simulation

To run a static simulation, access the Simulation Settings dialog by pressing **F2** or by selecting **Run | Simulation Settings…** from the main menu, and select

"Static Model" from the **Time Basis** drop-list. The **Duration** is then automatically set to 0:



You can then specify the number of realizations within the **Monte Carlo** tab of the dialog.

Read more: Setting the Monte Carlo Options for a Reliability Model (page 85).

Chapter 5: Advanced Reliability Modeling Concepts

> The first flight was relatively uneventful. Just one emergency, and another minor problem. A canopy-unsafe light illuminated at Mach 1.2 on the way to 1.5 at 50,000 feet, and later, ... fuel siphoning occurred. Not bad, as initial flights go.
>
> Robert Gilliland (describing the first flight of the SR-71 Blackbird, December 1964)

Chapter Overview

This chapter describes a number of advanced features that are available within the Reliability Module. These features allow for much more detailed representations of component behavior within a system.

In this Chapter

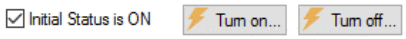
This chapter discusses the following topics:

- Turning Components On and Off in the Reliability Module
- Defining Failure Modes
- Failure Mode Control Variables
- Modeling Maintenance in the Reliability Module
- Modeling Resources in the Reliability Module
- Advanced Features of the Action Element

Turning Components On and Off in the Reliability Module

Function elements and Action elements both have buttons that provide access to triggers that allow you to turn them On or Off. In order for a Function or Action to be operable, it must be turned On.

On and Off triggers can be specified by clicking the **Turn on…** and **Turn off…** buttons in the reliability element property dialog:



Clicking either button brings up the standard triggering dialog. You can turn the element On or Off when an event occurs, when an expression or the output of another element changes, or when an expression becomes true or false.

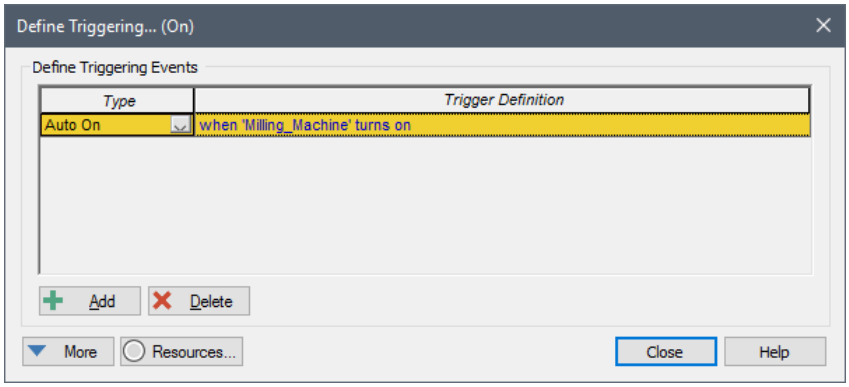
Read more: Defining Triggers (page 20).

By default, all reliability elements are "On" at the start of each realization. If you want the element to be "Off" at the start of each realization, you can simply clear the **Initial Status is ON** checkbox in the reliability element's property dialog.

On triggers also have an Auto On option. If the reliability element is not a child element inside a component being modeled as a system, the Auto On option triggers the element On when its parent Container is activated. This is only of use if the reliability element is within a conditional Container (since otherwise, the reliability element would be turned On at the start of the simulation).

Note: Conditional Containers are discussed in Chapter 10 of the **GoldSim User's Guide**.

If the reliability element is a child element inside a component being modeled as a system, the Auto On option triggers the element On when its parent component is triggered On:



Read more: Modeling a Reliability Element as a System with Child Elements (page 58).

Similarly, Off triggers have an Auto Off option. If the reliability element is not a child element inside a component being modeled as a system, the Auto Off option triggers the element Off when its parent Container is deactivated. Again, this is only of use if the reliability element is within a conditional Container

(since otherwise, the reliability element would never be turned Off during a simulation).

If the reliability element is a child element inside a component being modeled as a system, the Auto Off option triggers the element Off when its parent component is triggered Off.

Within the triggering dialog, you can also specify Resource Requirements necessary to turn the component On.

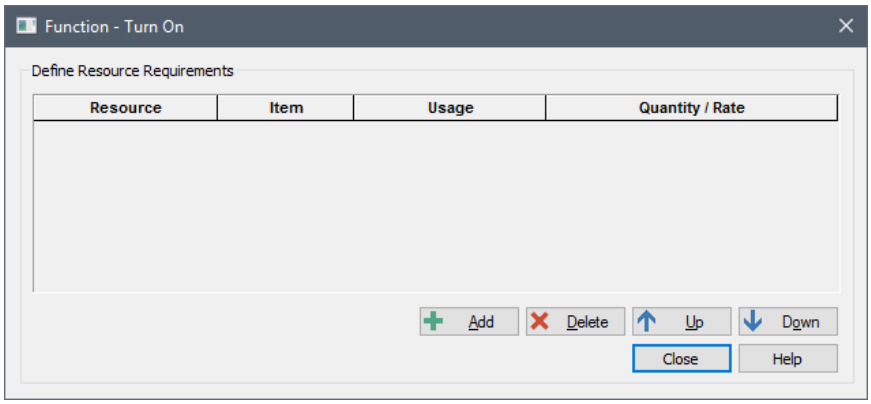
Specifying Resource Requirements for Turning Components On

A ***Resource*** is something that has a limited supply (e.g., spare parts, fuel, skilled personnel, money) and is required in order for elements of the modeled system to carry out certain actions.

Read more: Modeling Resources in the Reliability Module (page 121).

When specifying a trigger for turning a Function or an Action On, you can specify that certain Resources must be available in order for the component to be turned On. If the Resources are not available when the element is triggered to turn On, it will not turn On until they are available.

To define Resource Requirement for turning On a component, press the **Resources** button in the Turn On Trigger dialog. The following dialog will be displayed:



Note: In order for this dialog to appear, you must have previously defined at least one Resource in the model.

You can add a Resource Requirement by pressing the **Add** button.

The On trigger interacts with the specified Resource Stores when it is triggered, and can only have three types of interactions (specified in the **Usage** column):

- **Spend (discrete):** A discrete quantity of the Resource is required in order to complete the On trigger. If the requested Resource quantity is not available, the trigger signal is "held", and it waits for the Resource to become available.
- **Borrow (discrete):** A discrete quantity of the Resource is required in order to complete the On trigger. If the requested Resource quantity is not available, the trigger signal is "held", and it waits for the Resource to become available. If the Resource is available, the element is turned On and the borrowed quantity is returned to the Resource Store when the element is turned Off.

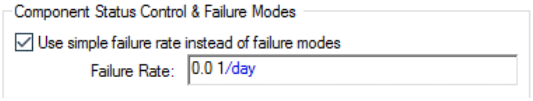
- **Deposit (discrete):** A discrete quantity of the Resource is created and deposited with the Store when the element is turned On.

Defining Failure Modes

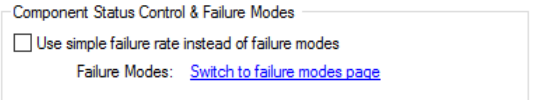
By default, when a new reliability element is created, a simple, unrepaired exponential failure mode is available (and it is assumed that this failure mode causes the component to stop operating when it occurs).

Note, however, that this is a very simple way to model failure (and therefore should be used with caution). An unrepaired, exponential failure (with a constant hazard rate) may be appropriate for random (and typically externally-generated) failures, but is not likely to be appropriate for failure due to processes such as wear and tear. Moreover, failure modes do not necessarily have to be fatal.

Fortunately, one of the Reliability Module's greatest strengths is its ability to represent failure modes in great detail. This is done by clearing the **Use simple failure rate instead of failure modes** checkbox:



When you do this, the **Failure Rate** input field is removed and replaced with a hyperlink:

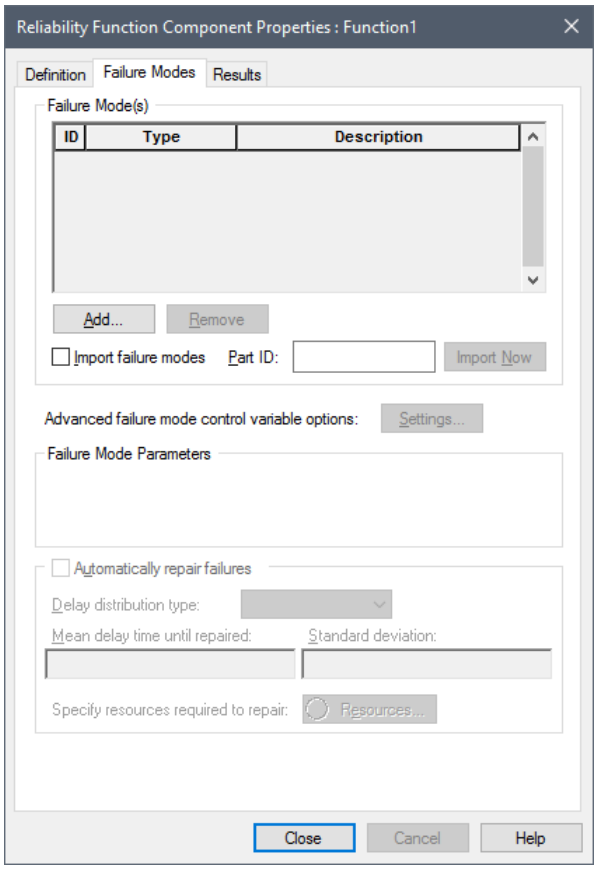


In addition, a **Failure Modes** tab is added to the dialog. Clicking on the tab or the hyperlink shown above displays the **Failure Modes** tab, which provides access to the Reliability Module's advanced failure mode features.

These advanced features are discussed in detail in the sections below.

The Failure Modes Tab

After clearing the **Use simple failure rate instead of failure modes** checkbox in a reliability element dialog, clicking on the **Switch to failure modes page** hyperlink or on the **Failure Modes** tab displays the **Failure Modes** tab:

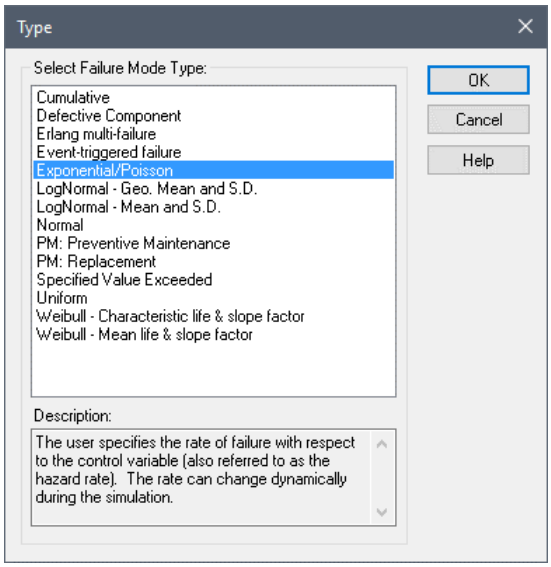


Note: If a non-zero **Failure Rate** was defined in the main dialog prior to activating the **Failure Modes** tab, this failure mode is automatically converted into the first failure mode in the **Failure Modes** tab.

Warning: If, after defining one or more failure modes, you subsequently check the **Use simple failure rate instead of failure modes** checkbox on the main dialog, all failure modes are permanently deleted.

Adding Failure Modes

GoldSim supports up to 10 independent failure modes for each reliability element. To add a failure mode, you click the **Add…** button within the **Failure Modes** tab of the element. The following dialog will be displayed:

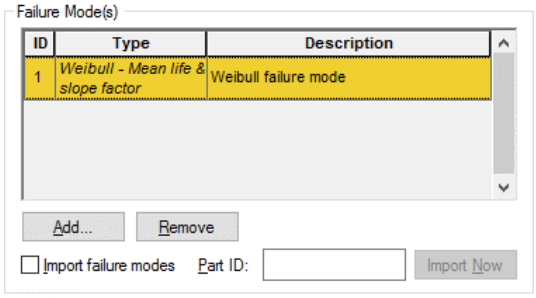


This dialog provides a list of failure mode types supported by the current reliability element type, along with a short description.

Note: Some failure mode types are only available for Action elements.

Read more: Failure Modes Available for Function and Action Elements (page 96); Failure Modes Available Only for Action Elements (page 99).

After selecting a failure mode type, a new failure mode will be added (and if previous modes existed, it will be added below the failure mode that was highlighted when the **Add** button is pressed):

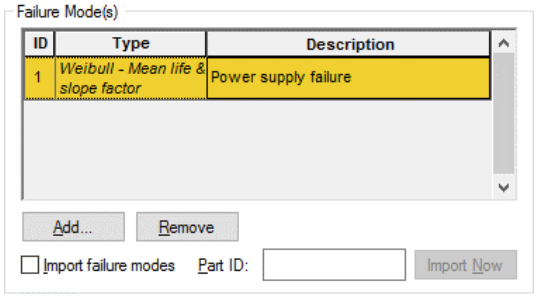


Note: By default, a numerical **ID** is assigned to each failure mode based on the order in which it was created. However, the IDs of failure modes can be changed to any integer between 1 and 10 (although each mode must have a unique ID).

The failure mode number is important because it appears as an Internal Requirement in the "Operating Requirements" portion of the Reliability element dialog. By default, each failure mode is added as a Not node with the argument ~Failed[n], where n is the failure mode number

Read more: Failure Modes and Internal Requirements (page 95).

Descriptive text can also be added to each of the failure modes (the default text is a description of the failure mode type):



Each failure mode type has one or more parameters (in the "Failure Mode Parameters" section of the dialog). To edit the parameters simply select the failure mode you wish to edit (by clicking on the **ID**, **Type** or **Description**). When you do so, the displayed Failure Mode Parameters (in the section below) will change to those of the selected failure mode. The input fields for Failure Mode Parameters can accept number, expressions and links from other GoldSim elements. They can also be specified as functions of time.

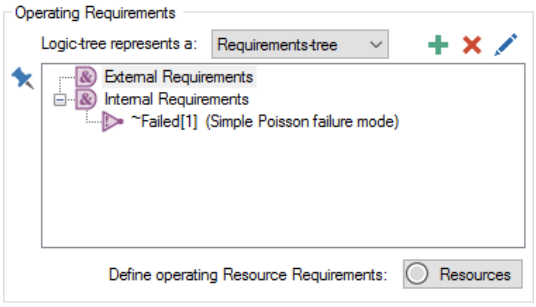
Read more: Changing Failure Mode Parameters Dynamically (page 100).

Failure Modes and Internal Requirements

By default, any failure modes that are added to a Reliability element are automatically added as Internal Requirements in the element's "Operating Requirements" tree. In particular, GoldSim automatically inserts a "Not" condition in the internal requirements portion of the requirements tree: Not ~Failed[n], where n is the failure mode.

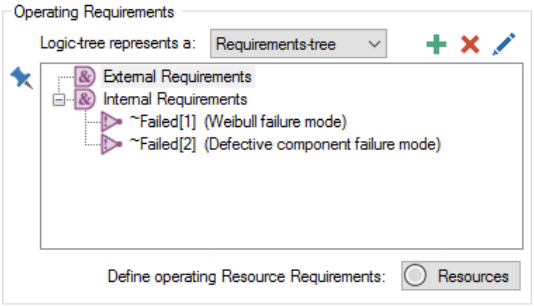
Read more: Defining Operating Requirements for Reliability Elements Using Logic Trees (page 64); Condition and Not Nodes (page 71).

For example, when you create a new Reliability element, it automatically has a single default failure mode. Note that Not ~Failed[1] automatically is added as an Internal Requirement:



The triangle with the ! inside represents a Not condition. The node is True if the condition (in this case, ~Failed[1]) is False.

If the element had two failure modes, the Operating Requirements would look like this:



What does this Internal Requirement refer to? Recall that the Reliability element provides an output called Failed that is a vector of ten items. It is based on an array label set named FailureModes (with ten items from 1 to 10) that is automatically provided by GoldSim. Each item in the vector corresponds to a failure mode defined for the element (GoldSim allows up to 10 failure modes). Failed[1] is a scalar output that corresponds to failure mode # 1. Each item is set to True if its corresponding failure mode has occurred (and it is reset to False if the mode is repaired).

Read more: Common Reliability Element Outputs (page 60).

When referenced inside the element itself, however, the Failed vector must be referenced as ~Failed. ~Failed is referred to as a locally available property.

Read more: Common Reliability Element Locally Available Properties (page 62).

Locally available properties derive their name from the fact that they may only be available, or they may take on different values (i.e., be over-ridden), in "local" parts of your model (e.g., within a particular element, or within a particular input field for an element). Locally available properties are referenced by prefacing them with the ~ operator. Since the Failed vector is being referenced within the element, it must be referenced in the Requirements tree as a local variable.

The triangle with the ! inside represents a **Not** node in the tree, so **Not ~Failed[1]** is True if failure mode #1 has not occurred and False if it has occurred. This implies that the failure mode is assumed to be fatal to the component (i.e., if the failure mode occurs, the component itself fails and is no longer operative). That is, by default, whenever you add a failure mode, GoldSim treats it as if it is a fatal failure mode.

If desired, however, you can specify that a failure mode is non-fatal. To do so, you simply need to remove the Not node from the Requirements tree.

Read more: Adding, Removing and Editing Nodes in the Logic-Tree (page 68).

In this case, you will most likely want to specify that the failure, while not fatal, reduces the capacity or performance of some internal component of the system by referencing **Failed[n]** in an ancillary calculation (e.g., throughput).

Read more: Modeling Coupled and Non-Fatal Failure Modes (page 105).

Failure Modes Available for Function and Action Elements

The failure modes types available for both the Function and Action elements are described below.

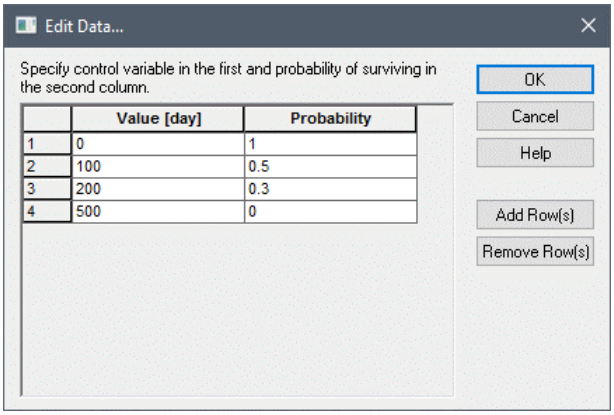
Note that most failure modes are defined relative to a ***failure mode control variable.*** This is the variable that is referenced by the failure mode to determine when failure occurs. (For those failure modes that are defined as distributions, the control variable represents the x-axis of a failure distribution plot.) By default, the failure mode control variable is the operating time since the

simulation began (or the last time the component was replaced). However, for any given failure mode, it can be set to the total time since the simulation began, the number of actions completed (available only for Action elements), or a user-defined variable (e.g., mileage). You can also accelerate or decelerate failure with respect to the control variable.

Read more: Failure Mode Control Variables (page 108).

Mathematical details of the failure modes are provided in Appendix A.

Cumulative. For this failure mode, you define a custom failure distribution by specifying a table of the (cumulative) failure mode control variable value and the probability of surviving. The dialog for defining this table is accessed via an **Edit…** button, and looks like this:



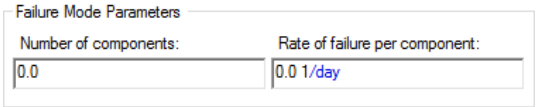
The table can only be defined using numbers (it does not accept expressions or links to other elements). You add and delete rows using the **Add Row(s)** and **Remove Row(s)** buttons.

Note: Probabilities must decrease monotonically as time increases.

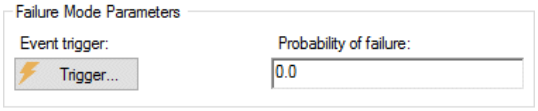
Defective Component. For this failure mode, you specify the probability that the component will be susceptible to the mode, and the (Poisson) failure rate if it is susceptible. As its name implies, it is used to simulate (typically rapid) failure due to a small fraction of defective components.

Erlang multi-failure. This failure mode is used to represent the failure of a number of identical sub-components, each of which fails according to the same Poisson failure rate, and is used to simulate a system that has N-1 spare parts, that are replaced immediately.

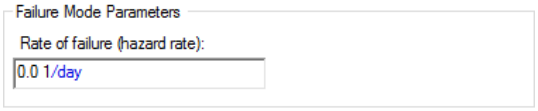
It is assumed that the first sub-component operates until it fails, at which time it is replaced by the second identical second sub-component, and so on until they have all failed. When all sub-components have failed, the component is assumed to have failed.



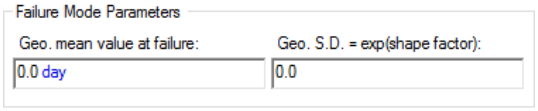
Event-triggered failure. For this failure mode, you specify a trigger (via a triggering dialog) and the probability of failing whenever the triggering event occurs:



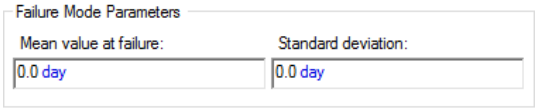
Exponential/Poisson. This is identical to the default failure mode if the Failure Mode tab is not used. You define an exponential failure distribution by specifying the rate of failure (also referred to as the hazard rate) with respect to the failure mode control variable:



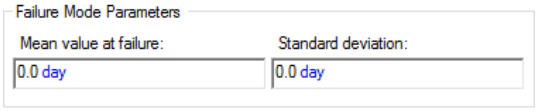
LogNormal. For this failure mode, you define a log-normal failure distribution with respect to the failure mode control variable. Two LogNormal failure mode types are provided. This allows you to define the distribution in terms of either a Geometric Mean and Geometric Standard Deviation:



or an arithmetic Mean and Standard Deviation:

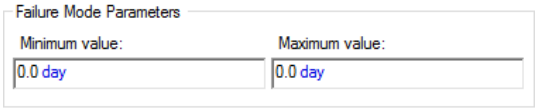


Normal. For this failure mode, you define a normal failure distribution with respect to the failure mode control variable. To do so, you specify a Mean and Standard Deviation:



Specified Value Exceeded. For this failure mode, you specify the value for the failure mode control variable which results in failure if it is exceeded:

Uniform. For this failure mode, you define a uniform failure distribution with respect to the failure mode control variable. To do so, you specify a Minimum value (lower bound) and a Maximum value (upper bound):



Weibull. For this failure mode, you define a Weibull failure distribution with respect to the failure mode control variable. Two Weibull failure mode types are provided. This allows you to define the distribution in terms of either a Mean life and Slope factor:

or a Characteristic life and Slope factor:

Note: The input fields for Failure Mode Parameters can accept numbers, expressions and links from other GoldSim elements. They can also be specified as functions of time. Depending on the failure mode type, however, parameters that are defined as a function of time may not change instantaneously (i.e., they may only change when the mode is repaired or the component replaced).

Read more: Changing Failure Mode Parameters Dynamically (page 100).

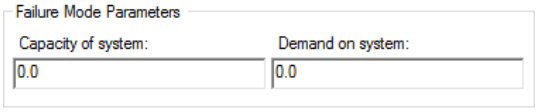
There are two additional failure mode types that are used to model preventive maintenance.

Read more: Modeling Maintenance in the Reliability Module (page 114).

Failure Modes Available Only for Action Elements

There are two failure modes types that are available only for Action elements.

Demand>Capacity. For this failure mode, you specify a Demand on the system (e.g., a load) and a Capacity of the system (e.g., a strength). If the demand exceeds the capacity when the action is triggered, failure occurs:



Typically, at least one of these two inputs would be defined using a Stochastic element, which should be triggered for resampling whenever the Action element is triggered.

Note: The Capacity and Demand must be entered as dimensionless values. However, theoretically, they will typically have dimensions. To represent these in your models, they should be entered with units, and then cast to dimensionless values (e.g., MassLimit|kg|). Casting dimensioned variables to dimesionless values is discussed in Chapter 3 ("Unit Casting") of the **GoldSim User's Guide**.)

Unreliable. For this failure mode, you specify the element's Reliability. This is a value between 0 and 1 that defines the (random) probability that it will successfully perform its action when it is triggered:

Note: The Unreliable failure mode does not actually cause the Action element to fail. Rather, it allows the user to specify the probability that an Action element's Action will be carried out successfully when it is triggered.

Changing Failure Mode Parameters Dynamically

The input fields for Failure Mode Parameters can accept numbers, expressions and links from other GoldSim elements. They can also be specified as functions of time.

Depending on the failure mode type, however, failure mode parameters that are defined as a function of time may not change immediately. In particular, some parameters are only changed when the mode is repaired or the component is replaced.

The following table categorizes the inputs for each failure mode according to whether changes are reflected immediately, or only after a failure mode has been repaired or the component replaced

Failure mode	Changes take effect immediately	Changes take effect when failure mode is repaired or component is replaced
Exponential/Poisson	Failure rate	
LogNormal		Mean (Geom. Mean); SD (Geom. SD)
Normal		Mean; SD
Uniform		Minimum value; Maximum value
Weibull		Mean life (Characteristic life); Slope factor
Specified Value Exceeded	Value at failure	
Event triggered failure	Probability of failure	
Defective component	Failure rate	Probability of defect
Erlang multi-failure		Failure rate; Number of components
Demand>Capacity	Demand; Capacity	
Unreliable	Reliability	

Note: The Cumulative failure mode does not allow you to change the inputs dynamically since the table must be specified using only numbers (and not equations or links).

Read more: Example: Modeling Dynamic Failure Mode Behavior Such as Burn-In (page 154).

Modeling the Repair of Failure Modes

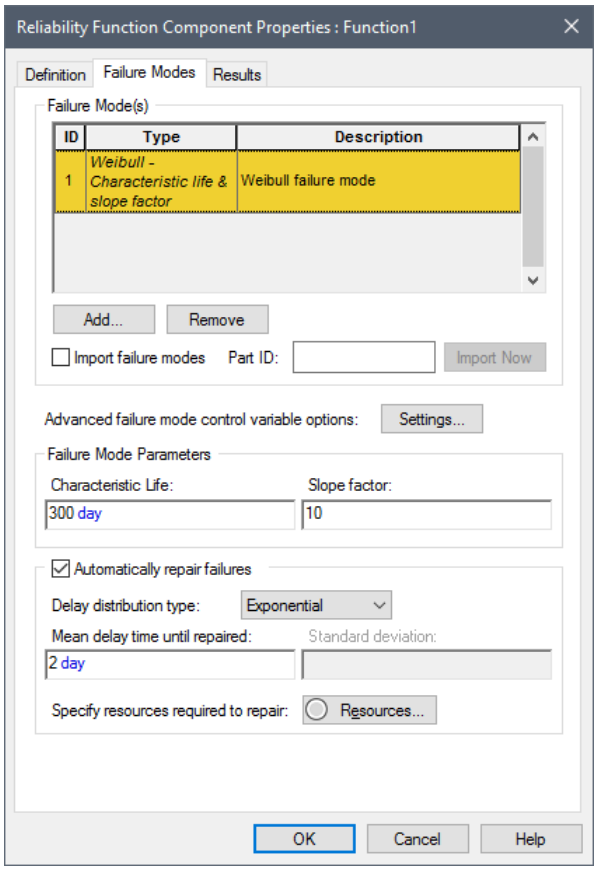
Each failure mode defined using the **Failure Modes** tab (with the exception of the Unreliable failure mode type for Action elements) can be specified to be repaired, with a separate repair time distribution.

Note: For the two PM (preventive maintenance) "failure modes", the repair time actually represents the amount of time required to complete the preventive maintenance.

Read more: Simulating Preventive Maintenance as a Failure Mode (page 114).

Repair is defined in terms of a repair time distribution, as well as specification of when the repair is to take place.

The repair time distribution is defined at the bottom of the **Failure Modes** tab:



GoldSim allows you to select from three probability distributions for the repair time:

- Gamma distribution;
- Lognormal distribution; and
- Exponential distribution.

All three repair distributions are defined by a mean delay time (until repaired) and a standard deviation must be specified for the first two types.

When a repair is triggered, GoldSim automatically samples from the specified repair time distribution to determine when the failure mode is repaired.

The parameters describing the repair distributions can be functions of time, but changes to the parameters only take effect when the next repair begins.

Note: If you specify a zero Mean delay time for the repair, GoldSim will treat this as follows: The time to implement the repair will be the lesser of 1 second or 1% of the length of the next timestep.

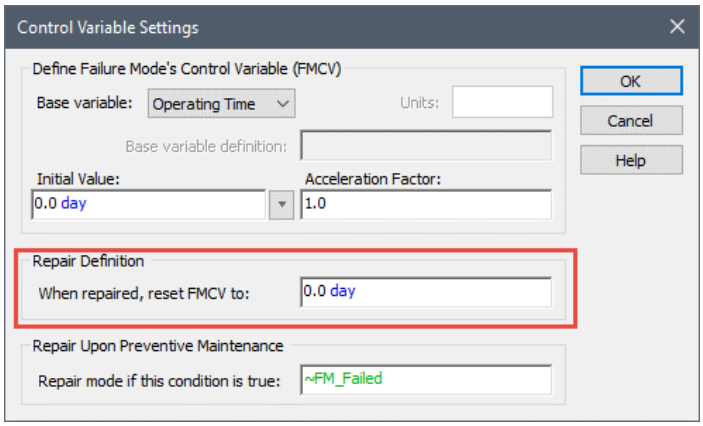
You can optionally specify that one or more Resources (e.g., spare parts, personnel) must be available in order to carry out the repair. If Resources are specified for the repair, the repair will not begin until the Resources are available.

Read more: Specifying Resource Requirements for Repairs (page 104).

It is important to understand exactly what is meant by a "repair". Most failure modes are defined relative to a failure mode control variable (e.g., the total time, operating time, mileage).

Read more: Failure Mode Control Variables (page 108).

For these types of failure modes, a repair means that the failure mode control variable (FMCV) has been reset back to a specific value. Selecting one of these failure modes and clicking the **Settings…** button in the **Failure Modes** tab provides access to the Control Variable Settings dialog. Within this dialog, you can define the **Repair Definition** for the mode:



By default, this value is zero, but it can be specified by the user to be any value.

For two failure modes (Demand>Capacity and Event-triggered failure), there is no failure mode control variable. Hence, repair simply means that the failure mode is no longer making the component inoperable (since there is no failure mode base variable to reset).

Note: Although Exponential/Poisson and Defective Component failures are FMCV-based, they are memoryless, since their hazard function is constant with time. As a result, the value to reset the FMCV to cannot be specified for these failure modes (as it would have no meaning).

As pointed out above, in addition to specifying a repair time distribution, you must also specify *when* the repair is triggered to take place. That is, you must explicitly specify what triggers a repair to begin.

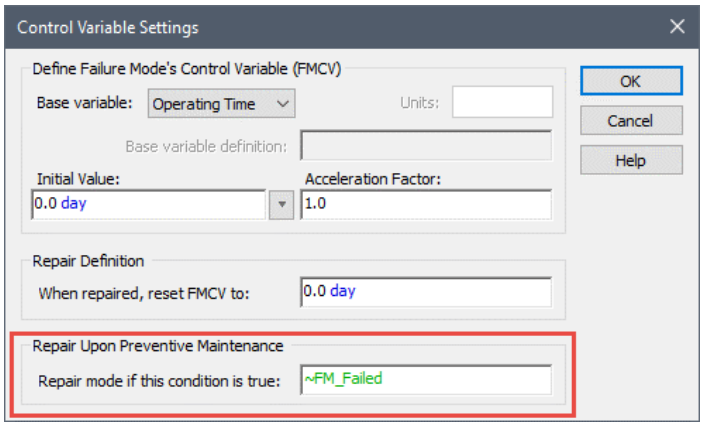
Repairs of failure modes can triggered in two different ways:

- They can be repaired automatically, starting as soon as they fail; and
- They can be repaired when a PM (preventive maintenance) is carried out.

You control whether one, none or both of these trigger a repair.

To automatically repair failures for a specific failure mode, select the failure mode, and then check the **Automatically repair failures** checkbox. This forces the repair to begin immediately upon failure.

You can also specify that a failure mode can be repaired when a preventive maintenance is carried out, even if it has not yet failed. You do so through the Control Variable Settings dialog (accessed, as mentioned above, by selecting a failure mode and pressing the **Settings…** button in the **Failure Modes** tab). From this dialog, you can specify whether or not the mode is repaired upon preventive maintenance:



If this input field is true, the repair of the failure mode begins when the PM mode begins. Otherwise, it is not repaired.

By default, this condition field is initialized to

~FM_Failed

FM_Failed is a locally available property of the failure mode. It is true if the failure mode is in a failed state, and false if the failure mode is in an unfailed state. Hence, by default *a Preventive Maintenance event automatically repairs other failure modes that are currently in a failed state, but does not repair other failure modes that are in an unfailed state.*

Note: When specifying whether or not a PM repairs a failure mode, another locally available property can also be referenced in the **Repair mode if this condition is true** field: FM_TimeToFail. This variable represents an *estimate* of the time to failure (if the mode is already failed, it will be 0). This allows you to specify that the mode is to be repaired even if it has not yet failed, based on your prediction that it will likely fail soon.

Read more: Simulating Preventive Maintenance as a Failure Mode (page 114).

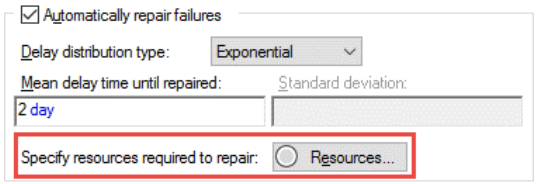
Specifying Resource Requirements for Repairs

A ***Resource*** is something that has a limited supply (e.g., spare parts, fuel, skilled personnel, money) and is required in order for elements of the modeled system to carry out certain actions.

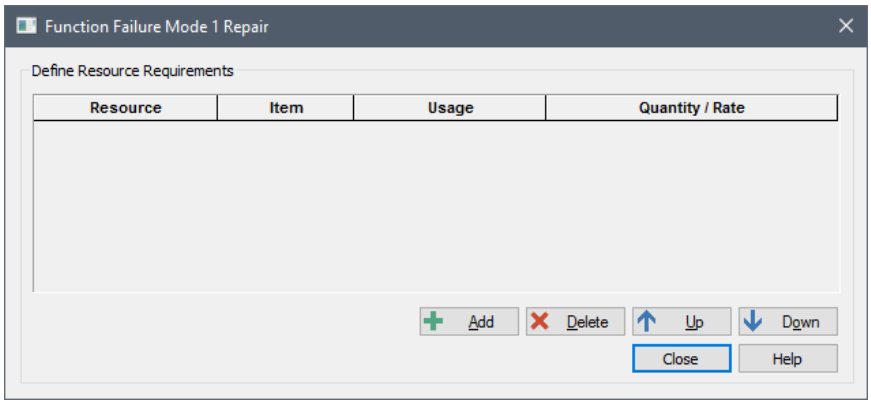
Read more: Modeling Resources in the Reliability Module (page 121).

When automatically repairing a failure mode, you can specify that certain Resources must be available in order for the failure mode to be repaired. If the Resources are not available when the component fails due to that mode, the mode will not be repaired until they are available.

To define Resource Requirements for repairing a failure mode, press the **Resources** button at the bottom of the **Failure Modes** tab:



The following dialog will be displayed:



Note: In order for this dialog to appear, you must have previously defined at least one Resource in the model.

You can add a Resource Requirement by pressing the **Add** button.

An repair for a failure mode interacts with the specified Resource Stores when the component fails due to that mode, and can only have two types of interactions (specified in the **Usage** column):

- **Spend (discrete):** A discrete quantity of the Resource is required in order to carry out the repair. If the requested Resource quantity is not available, the repair signal is "held", and it waits for the Resource to become available.
- **Borrow (discrete):** A discrete quantity of the Resource is required in order to carry out the repair. If the requested Resource quantity is not available, the repair signal is "held", and it waits for the Resource to become available. If the Resource is available, the repair begins and the borrowed quantity is returned to the Resource Store when the repair is complete.

Modeling Coupled and Non-Fatal Failure Modes

By default, any failure modes that are added to a Reliability element are automatically added as Internal Requirements in the element's "Operating Requirements" tree. In particular, GoldSim automatically inserts a "Not" condition in the internal requirements portion of the requirements tree: Not ~Failed[n], where n is the failure mode.

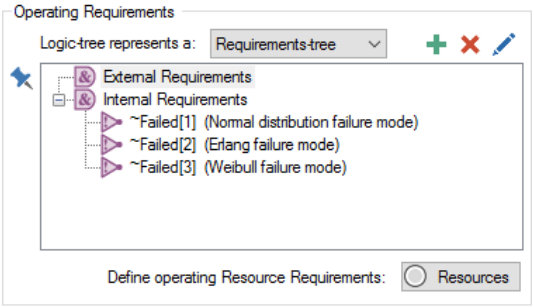
Read more: Failure Modes and Internal Requirements (page 95).

This implies that the failure mode is assumed to be fatal to the component (i.e., if the failure mode occurs, the component itself fails and is no longer operable). That is, by default, whenever you add a failure mode, GoldSim treats it as if it is a fatal failure mode.

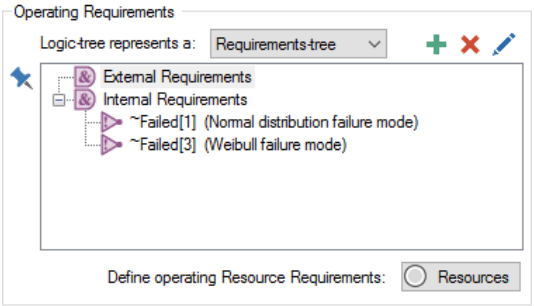
In some cases, however, the failure mode may not be fatal to the component (i.e., when the failure mode occurs, it does not cause the component itself to fail). This type of situation can be modeled by simply removing references to non-fatal failure modes from the Requirements tree.

Read more: Adding, Removing and Editing Nodes in the Logic-Tree (page 68).

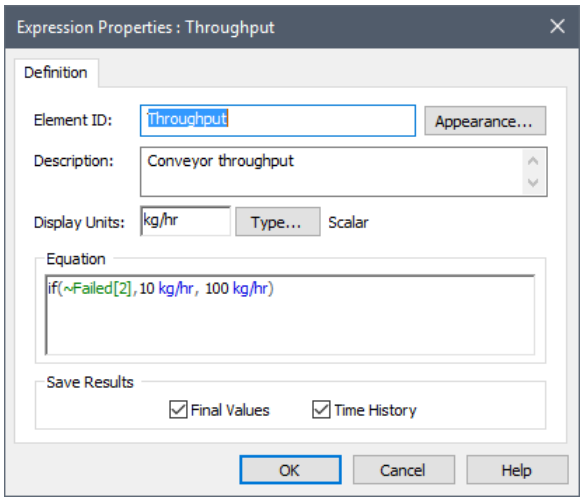
For example, let's consider a Reliability element with three failure modes. By default, when we are done adding the three failure modes, the Requirements tree will look like this:



If you wanted to make the second failure mode non-fatal, you would simply select the node and press the Remove button (the red X). This would mean that the component would continue to operate if the second failure mode occurred. The Requirements tree would then look like this:



In this case, you would most likely want to specify that the failure, while not fatal, reduces the capacity or performance of some internal component of the system by referencing **Failed[n]** in an ancillary calculation. For example, inside the reliabililty element (that was modeled as a system), you may have an equation defining a throughput rate that looks like this:



Note that in this case, the failure is referenced as a local variable, since it is internal to the Reliability element.

Read more: Failure Modes and Internal Requirements (page 95).

In other cases, you may want to specify that more than one failure mode must occur to cause the component to stop operating. In this case, you can use gate nodes (an OR node in this example) to customize the failure logic.

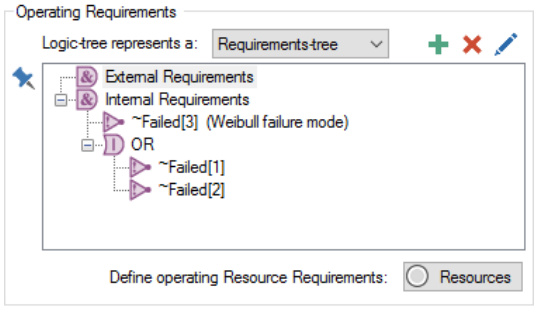
Again, let's consider the case where a component has three failure modes. Let's assume, however, that the component fails if either 1) the third failure mode occurs, or 2) both the first and the second failure modes occur.

To represent this, we would do the following:

1. Remove the first and second failure modes from the Internal Requirement list.
2. Add an Or node.
3. Under the Or node, add a Not node with the argument ~Failed[1].
4. Under the Or node, add a Not node with the argument ~Failed[2].

Read more: Understanding Logic Tree Nodes (page 68); Condition and Not Nodes (page 71).

The requirements tree would then look like this:



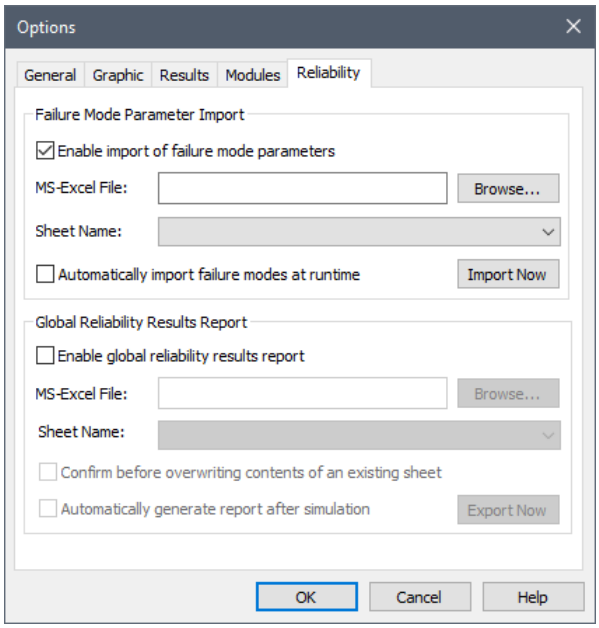
It indicates that the requirement for the component to operate is that failure mode #3 has not occurred AND that either failure mode 1 OR failure mode 2 has not occurred.

Read more: Example: Modeling Non-Fatal Failure Modes (page 161).

Importing Failure Mode Information from Spreadsheets

GoldSim provides the option to import failure mode information from Microsoft Excel into Reliability elements. When the feature is enabled, GoldSim is able to import details of individual failure modes (including descriptions and repair information) from a user-specified spreadsheet.

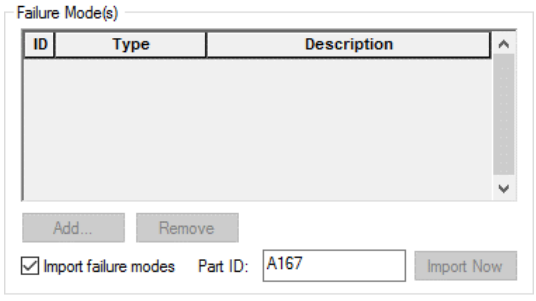
To enable the feature, select **Model|Options…** from the main menu, click on the **Reliability** tab of the dialog, and check the **Enable import of failure mode parameters** option:



You must then specify an Excel spreadsheet filename and the worksheet where failure mode data is stored. The required spreadsheet format is described in Appendix B, and is illustrated in a template spreadsheet file FailureImport.xlsx (found in the Reliability Examples folder in your GoldSim directory).

Read more: Appendix B: Failure Mode Import Spreadsheet Format (page 195).

Once Failure Mode import has been enabled globally, the feature must also be enabled for those Reliability elements for which you wish to import failure distributions. To do this, you must check the **Import failure modes** box and specify a **Part ID** listed in the failure modes spreadsheet:



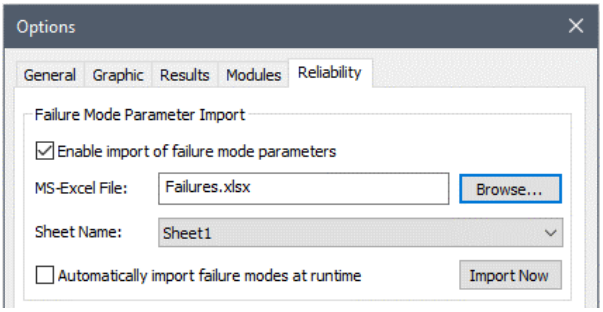
Note: Multiple elements will often have the same Part ID (e.g., five pumps all of the same type). Moreover, within the spreadsheet, Part IDs will often have multiple failure modes specified.

Note: Part IDs are not case-sensitive.

Note: If the Import failure modes box is checked, you cannot manually edit, Add or Remove failure modes.

You can import failure modes individually for each element by pressing the **Import Now** button in the Failure Modes dialog. When you do so, all of the failure modes for that Part ID from the specified spreadsheet will be imported for the element.

You can also simultaneously import the failure modes for all Reliability elements for which failure mode import has been activated by using the **Import Now** button in the **Reliability** tab of the **Model|Options…** dialog:



Finally, if you check the **Automatically import failure modes at runtime** option in the **Reliability** tab of the of the **Model|Options** dialog, GoldSim will automatically update failure mode data whenever the model is placed into Ready mode.

Note: When you import failure modes, any failure mode settings that are not explicitly provided in the spreadsheet (i.e., Resource requirements and advanced settings) are retained, as long as the Failure Mode ID and the Failure Mode Type are unchanged.

Warning: When you import failure modes, any existing non-PM failure modes are deleted. PM failure modes are kept if and only if they do not share a Failure Mode ID with any of the imported modes. Hence, if you do not wish you PM failure modes to be deleted when you import failure modes from a spreadsheet, you must ensure that the Failure Mode IDs for the PM failure modes do not conflict with any Failure Mode IDs in the spreadsheet.

Failure Mode Control Variables

Most failure modes are defined relative to a failure mode control variable (FMCV). This is the variable that is referenced by the failure mode to determine when failure occurs. (For those failure modes that are defined as distributions, the control variable represents the x-axis of a failure distribution plot.)

Note: Demand>Capacity, Event-triggered failure, and Unreliable failure modes are the only failure modes that are not defined relative to a failure mode control variable (i.e., they are not FMCV-based).

For FMCV-based failure modes, the failure mode calculates the FMCV "age" that will result in its next failure at the start of the realization, after each time the mode is repaired, and after each time the component is replaced. When the FMCV exceeds this value, the failure occurs.

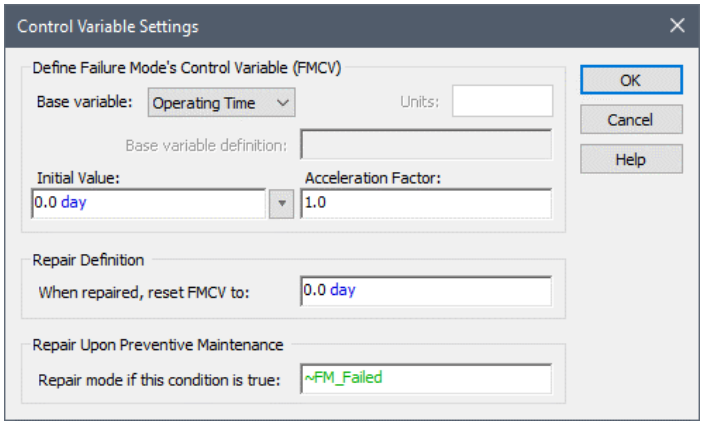
Each (FMCV-based) failure mode has its own FMCV, which is defined in terms of a specified "base variable" (e.g., total simulation time, operating time, mileage). The FMCV for a particular failure mode and the base variable are related as follows:

$$\text{FMCV(t)} = \text{Initial Value} + \sum_{i=1}^{\text{Nsteps}} (\text{Base Variable}_i - \text{Base Variable}_{i-1}) \ \text{Acceleration}_i$$

FMCV(t) represents the value for the FMCV at time t. The sum is made over all the timesteps from the beginning of the realization (or the time the FMCV was reset) to time t. The Initial Value and Acceleration Factor are user inputs for each failure mode. The FMCV calculation can be reset by certain events such as replacement of the component or repair of the failure mode. This is done by resetting the Initial Value (to a specified FMCV value) and restarting the sum.

For example, if you were simulating the wear-out failure mode for a brake system in a car, the Base Variable would be mileage, and the Acceleration factor could change dynamically (from 1) depending on whether the car was being used in the city or on the highway. The FMCV calculation would be restarted (by resetting the sum the Initial Value to 0) whenever the brakes were replaced.

Each failure mode's FMCV properties can be accessed from within the Failure Modes tab by selecting the failure mode and pressing the **Settings…** button. This displays the following dialog:



Failure mode control variables are discussed in detail below.

Understanding Failure Mode Base Variables

Each (FMCV-based) failure mode has its own FMCV, which is defined in terms of a specified "base variable". GoldSim provides four options for base variables:

- **Operating time**. This is the default base variable for any new failure mode. This base variable accrues time only when the element is Operating.

- **Total time**. FMCVs using this base variable accrue time from the in-service time of the component, regardless of the element's status. Total time is useful for modeling failure modes which are independent of operational status, and also for modeling the degradation of standby and backup systems.
- **Number of actions completed**. This base variable is only available to failure modes for Action elements. This base variable tracks the number of times the action has been triggered and has issued an ActionOK or ActionFailed event.
- **User-defined**. In some cases, it may be appropriate to define an FMCV which is defined with respect to a base variable other than time or the number of actions completed (e.g., mileage). Any monotonically increasing function can be specified as a base variable. In most cases, the user-defined base variable will be defined using an Integrator, Reservoir or Pool element.

It is important to note that when using the base variables that are based on time (e.g., total time and operating time), GoldSim can predict the occurrence of the next event (based on the values from the model's last update at a timestep or event) and will interrupt a timestep for that event. This means that failure modes with static **Acceleration Factors** are insensitive to the timestep length, at least from a perspective of how the system will evolve. However, you should note that the condition of the system is only recorded at scheduled timesteps.

Read more: Example: Understanding the Differences Between Failure Mode Base Variables (page 149).

Creating User-Defined Base Variables

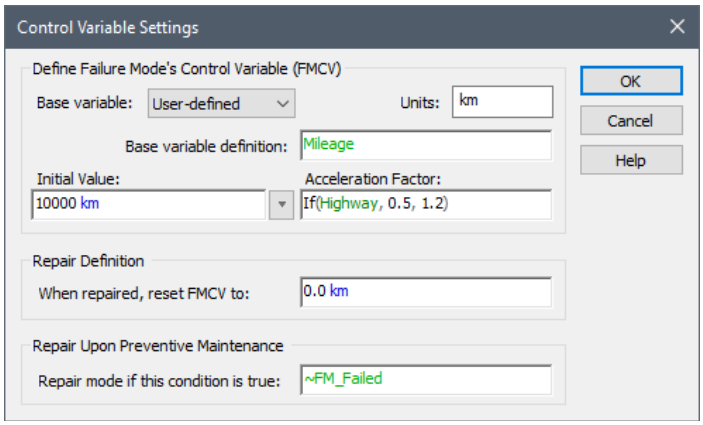
In some cases, it may be appropriate to define an FMCV which is defined with respect to a base variable other than time or the number of actions completed (e.g., mileage). Any monotonically increasing function can be specified as a base variable. In most cases, a user-defined base variable will be an Integrator, Reservoir or Pool element.

Warning: If your user-defined base variable does not increase (or remain unchanged) with time, GoldSim will display a fatal error message.

Note: Integrator, Reservoir and Pool elements (collectively referred to as Stock elements) are discussed in detail in Chapter 4 of the **GoldSim User's Guide**.

For example, if you were modeling the reliability of a car, you might want to use the car's mileage (i.e., distance traveled) the base variable. An Integrator element could be used to track the car's mileage through its life.

To make mileage a user defined FMCV, you would select **User defined** from the drop down list, specify miles as the **Units**, and simply create a link to the Mileage integrator in the **Base variable definition** field:



This failure mode's input parameters (e.g., mean and SD if it was a Normal failure mode) can now be defined in terms of distance (e.g., miles).

Note: In the example shown above, an Initial Value and an Acceleration Factor have been defined. These convert the base variable (which is independent of the failure mode) to an FMCV, which is specifically applied for this particular failure mode.

Read more: Specifying the Initial Value for Failure Mode Control Variables (page 111); Modeling Acceleration for Failure Mode Control Variables (page 112).

Timestep selection is important when using a User-defined base variable. This is because the value of the base variable is only recalculated at each timestep (unless an event occurs).

For example, say you are using a 1 month timestep, and the value of the FMCV increases by 2000 each month. If you have a component with an "age" with respect to the FMCV of 5000 at month 10, and a failure is specified to occur when the FMCV equals 5001, that failure should occur almost immediately after month 10. But if no event occurs during the month, the failure mode's "age" will not change until the next timestep, where its value will be 7000, causing failure. In this scenario, this means that the failure could occur nearly a month later than would be expected. A shorter timestep would mitigate this.

Read more: Example: Creating User-Defined Base Variables (page 151).

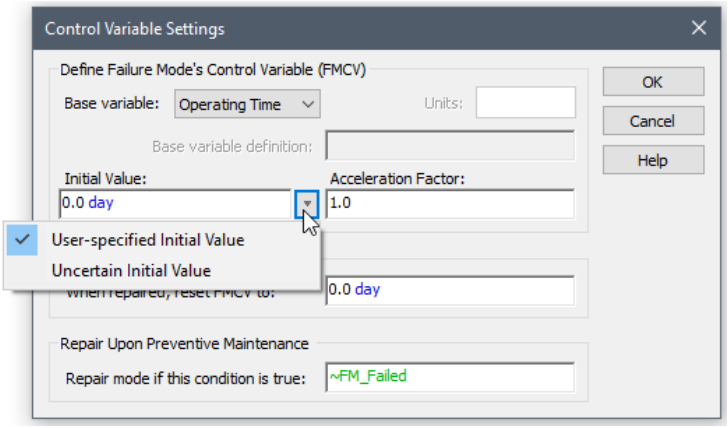
Specifying the Initial Value for Failure Mode Control Variables

Base variables are independent of failure mode (i.e., they are a property of the Function or Action element). FMCVs, however, are specific to each of the element's failure modes. The FMCV for a particular failure mode is computed by applying a specified Initial Value and an Acceleration Factor to the base variable.

That is, these two parameters convert the base variable (which is independent of the failure mode) to an FMCV, which is specifically applied for each particular failure mode. The Initial Value field is available for FMCV-based failure modes (except Exponential/Poisson).

Note: Although Exponential/Poisson and Defective Component failures are FMCV-based, they are memoryless, since their hazard function is constant with time. As a result, the Initial Value cannot be specified for these failure modes (as it would have no impact).

The Initial Value (which defaults to 0) allows you to model failure modes in which components are not new at the start of the simulation. There are two options for the Initial FMCV value, which are accessible via a drop-list:



If you know the current "age" of the component, it can be directly specified with the **User-specified Initial Value** option. If the current age of the component is unknown, you can use the **Uncertain Initial Value** option. In this case, GoldSim will randomly select an "age" between "new" and the value of the FMCV at the next failure of that failure mode.

Note: Although the **Initial Value** can be defined with an expression or link to another element's output (and hence could be specified to be a function of time), it is only evaluated at the start of each realization. When the FMCV is reset (due to repair or replacement), it is always reset to the specified Repair Definition, not the Initial Value

Note: Some confusion could potentially occur with regard to Initial Values when creating user-defined base variables. This is because user-defined base variables are likely to be Integrators or Reservoirs, which themselves have Initial Values. With regard to the FMCV, the Initial Value of the base variable itself (i.e., the Integrator or Reservoir) is never used. GoldSim only monitors changes in the base variable and uses the Initial Value specified in the Failure Mode Control Variable dialog to determine the actual value of the FMCV.

Read more: Creating User-Defined Base Variables (page 110).

Modeling Acceleration for Failure Mode Control Variables

Base variables are independent of failure mode (i.e., they are a property of the Function or Action element). FMCVs, however, are specific to each of the element's failure modes. The FMCV for a particular failure mode is computed by applying a specified Initial Value and an Acceleration Factor to the base variable.

That is, these two parameters, along with the base variable (which is independent of the failure mode), are used to compute the FMCV, which is specifically applied for each particular failure mode:

$$\text{FMCV(t)} = \text{Initial Value} + \sum_{i=1}^{\text{Nsteps}} (\text{Base Variable}_i - \text{Base Variable}_{i-1})\ \text{Acceleration}_i$$

FMCV(t) represents the value for the FMCV at time t. The sum is made over all the timesteps from the beginning of the realization (or the time the FMCV was reset) to time t.

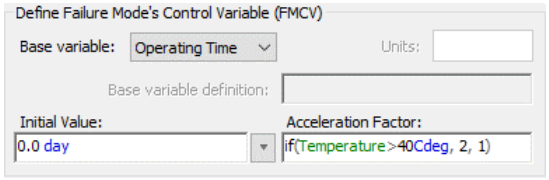
Hence, the **Acceleration Factor** field is available for all FMCV-based failure modes.

The acceleration factor is a non-negative real number which multiplies the actual change in the base variable to arrive at the failure mode's current "age". Setting this value to a number less than one means the component will age slower than normal (failure is decelerated), and setting it to a number greater than one will cause the component to age faster than normal (failure is accelerated).

Read more: Specifying the Initial Value for Failure Mode Control Variables (page 111).

For example, we might have a component which ages twice as fast when it operates in ambient temperatures of greater than 40 degrees Celsius.

To represent this, we would specify the following expression in the **Acceleration Factor** field:



The **Acceleration Factor** can also be used to predict how a system will perform in accelerated testing, or used along with accelerated failure data to predict the performance of a system in a regular duty cycle.

While the acceleration factor can be a function of time, it is important to note that the acceleration factor is only updated when the model is updated (model updates occur at timesteps and when certain events occur).

Acceleration factors will often be defined as functions of locally available properties such as operating time or total time. For example, you may want to accelerate failure after the component has been operating for 1 month. These variables are available as locally available properties within the Acceleration Factor input field.

Read more: Common Reliability Element Locally Available Properties (page 62).

Note: Although exponential failure modes are memoryless (i.e., they have a constant hazard function), they do support the use of acceleration. When used with an Exponential/Poisson failure mode, the specified failure rate is multiplied by the acceleration factor. So if an acceleration factor of 3 is specified, and a failure normally occurs 5 times per year, GoldSim will use a failure rate of 15 yr-1.

Read more: Example: Modeling Changing Operational Environments Using Failure Mode Acceleration (page 157).

Referring to FMCVs When Defining Failure Mode Parameters

In some cases, you may want to define the parameters of a failure mode using an expression that refers to the current value of the FMCV. When specifying failure mode parameters, the current "age" of the selected failure mode is available as a locally available property called FMCV (referenced as "~FMCV").

Read more: Common Reliability Element Locally Available Properties (page 62).

Note, however, that the FMCV local variable is only available for failure mode parameters that are "dynamic" (i.e., where changes take effect immediately).

Read more: Changing Failure Mode Parameters Dynamically (page 100).

Modeling Maintenance in the Reliability Module

GoldSim has the capability to simulate maintenance of your component. There are two types of maintenance that you can model in GoldSim:

- **Preventive maintenance** can be used to repair failure modes and set their FMCVs (i.e., their "age") back to a specified value. Preventive maintenance is simulated by defining one or more "PM failure modes".

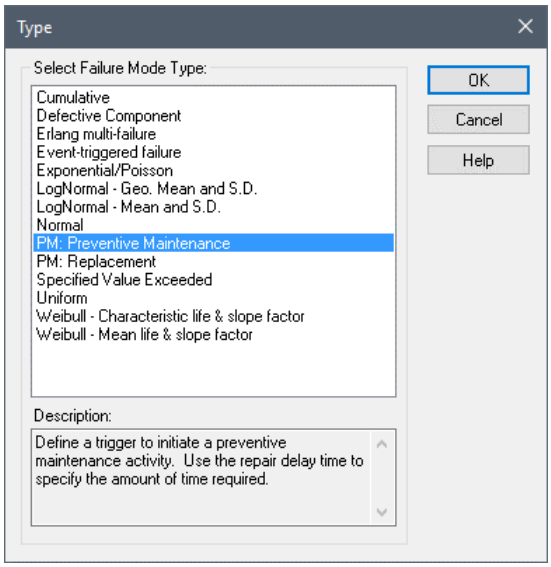
  By default, any failure mode that is in a failed state is automatically repaired by PM events (although you can override this for any given failure mode and indicate that it is not to be repaired). You can also specify whether PM events repair unfailed failure modes (by resetting their FMCV). By default, this does not occur.

- **Replacement** automatically repairs all failure modes and sets their FMCVs to zero (i.e., new). By definition, failure modes are always modified by the replacement (and you cannot override this). Replacement is simulated by either defining one or more "Replacement failure modes", or by directly triggering a replacement event (using the Replace trigger in an element's main properties page).

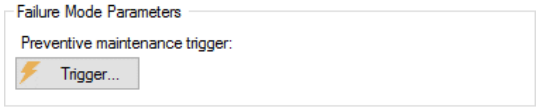
Modeling maintenance is discussed in detail in the sections below.

Simulating Preventive Maintenance as a Failure Mode

A Preventive Maintenance failure mode is added (and removed) in exactly the same manner as for other failure modes. The failure mode to select is **PM: Preventive maintenance**:



When you select a PM: Preventive Maintenance failure mode, you must then specify a trigger for the PM event (via a triggering dialog):



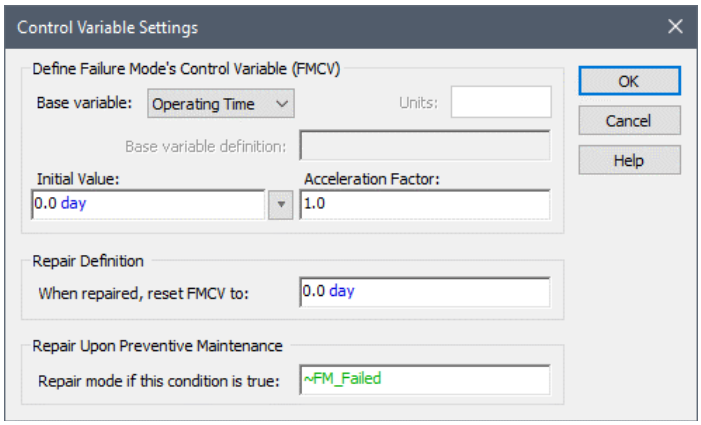
Read more: Defining Triggers (page 20).

As is the case for other failure modes, the Preventive Maintenance mode can be "repaired" automatically, using the same time distributions available to other failure modes. This does not mean a Preventive Maintenance mode is "repaired" in the same sense as other failure modes. Rather, it provides a way to specify the amount of time required to complete the preventive maintenance.

Note: You must select **Automatically repair failures** when defining a PM failure mode. Otherwise, the PM will continue indefinitely (and GoldSim will issue a warning message when the model is run).

Read more: Modeling the Repair of Failure Modes (page 101).

Whether or not a Preventive Maintenance event impacts another failure mode is determined by the Control Variable Settings for the other failure mode (accessed by selecting the failure mode in the **Failure Modes** tab, and pressing the **Settings…** button):



At the bottom of this dialog is a section labeled "Repair Upon Preventive Maintenance". This setting, in conjunction with the "Repair Definition" directly above, determines how the failure mode is impacted by a Preventive Maintenance event:

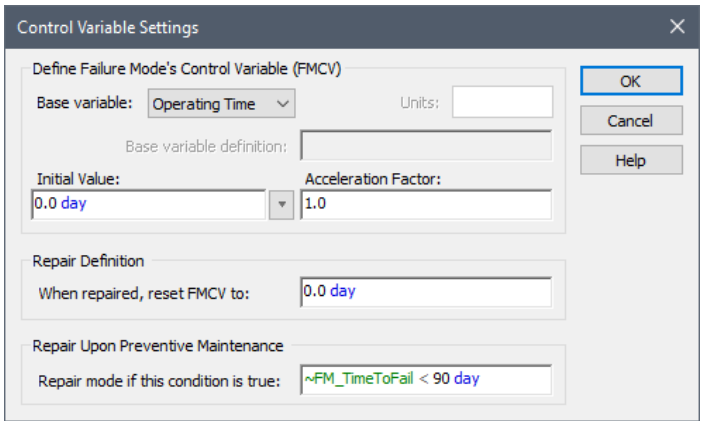
> **Repair mode if this condition is true**: If this input field is true, the failure mode repair begins when the PM mode starts. Otherwise, it is not repaired. If the failure mode is FMCV-based, the FMCV is set to the value specified in the "Repair Definition" when it is repaired.
>
> By default, this condition field is initialized to
>
> ~FM_Failed

FM_Failed is a locally available property of the failure mode. It is true if the failure mode is in a failed state, and false if the failure mode is in an unfailed state. Hence, by default *a Preventive Maintenance event automatically repairs other failure modes that are currently in a failed state, but does not repair other failure modes that are in an unfailed state.*

Note: The FM_Failed locally available property can only be used in the "Repair Following Preventive Maintenance" (**Repair mode if this condition is true**) field and the "Repair Definition" (**When repaired, reset FMCV to**) field. It is quite useful, as it allows you to use logic to differentiate the impact of a PM event on failed and unfailed modes.

Read more: Common Reliability Element Locally Available Properties (page 62).

When specifying whether or not a PM repairs a failure mode, another locally available property can also be referenced in the **Repair mode if this condition is true** field: FM_TimeToFail. This variable represents an *estimate* of the time to failure (if the mode is already failed, it will be 0). This allows you to specify that the mode is to be repaired even if it has not yet failed, based on your prediction that it will likely fail soon. For example, if you thought it would fail before the next PM (scheduled for 90 days from now), you could specify that the mode be repaired:



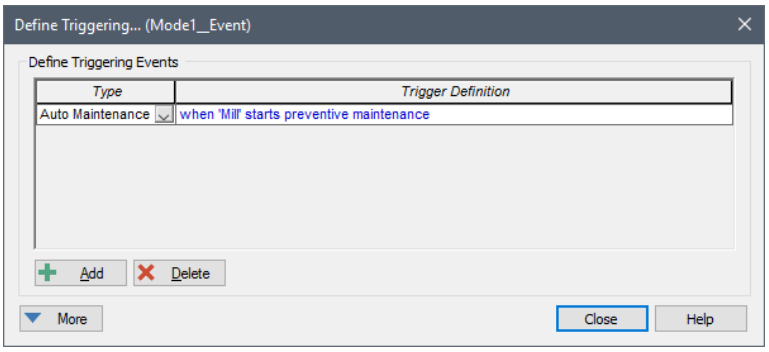
Using the FM_TimeToFail locally available property in this manner provides a mechanism by which you can use GoldSim to design and optimize a reliability-centered (or predictive) maintenance program. In particular, you can simulate alternative maintenance programs, and quantify the benefit of each. The concept is that during each PM, some diagnostic tests are carried out in order to estimate the time to failure. If this is below a predefined limit, the component is replaced or repaired rather than running the risk of failure while the system is operating.

Note: The FM_TimeToFail locally available property can only be used in the "Repair Upon Preventive Maintenance" (**Repair mode if this condition is true**) field.

Note: For some types of failure modes (e.g., Exponential/Poisson, Defective Component), it is not possible to compute an estimated time to failure. In these cases, FM_TimeTo_Fail is not provided as a locally available property.

When a reliability element that is being modeled as a system undergoes preventive maintenance, the impact on the child elements within the component is as follows:

- If a child element has no defined Preventive Maintenance modes of its own, it automatically undergoes preventive maintenance when its parent undergoes preventive maintenance.
- If a child element has at least one Preventive Maintenance mode of its own defined, it *does not* automatically undergo preventive maintenance when its parent undergoes preventive maintenance. However, an "Auto Maintenance" trigger will be available within the trigger dialog for the child's PM failure modes which can cause the child's PM mode to be triggered when the parent's PM mode is triggered:



Read more: Modeling a Reliability Element as a System with Child Elements (page 58).

Preventive maintenance triggers will often be defined as functions of locally available properties such as operating time or total time. For example, you may want to trigger a preventive maintenance every month of operating time. These variables are available as locally available properties within the trigger input field.

Read more: Common Reliability Element Locally Available Properties (page 62); Example: Modeling Component Maintenance and Replacement (page 158).

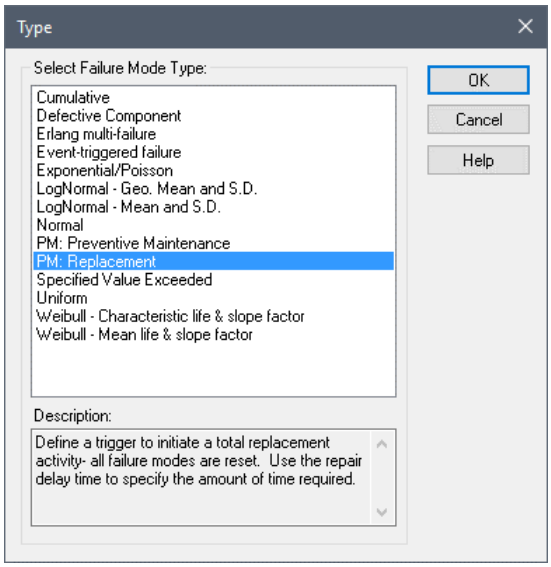
By default, like all failure modes, PM failure modes are added to the Internal Requirements list. As a result, when the PM is taking place, the element will be inoperable (and the Status for the element will report 1). However, you can choose to remove the PM failure mode from the Internal Requirements list. If you do so, when the PM is taking place, the element will be not be inoperable (and the Status for the element will NOT report 1; if all other requirements are met, for example, it would report 0).

Read more: Common Reliability Element Outputs (page 60); Failure Modes and Internal Requirements (page 95).

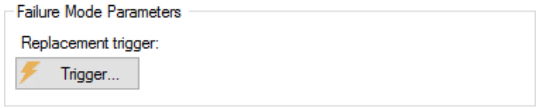
Note: When a PM takes place, any other failure mode that is repaired by the PM is automatically set to failed and a repair begins. This is important to keep in mind since as a result of this, when a PM occurs, the Status for the element will not necessarily report 1. If a triggered repair takes longer than the PM itself, the Status will first report 1, and then 2 (indicating a failed mode that has not yet been repaired), and then eventually 0 (when all modes are repaired).

Simulating Replacement as a Failure Mode

A Replacement failure mode is added (and removed) in exactly the same manner as for other failure modes. The failure mode to select is **PM: Replacement**:



When you select a PM: Replacement failure mode, you must then specify a trigger for the Replacement event (via a triggering dialog):



Read more: Defining Triggers (page 20).

As is the case for other failure modes, the Replacement mode can be "repaired" automatically, using the same time distributions available to other failure modes. This does not mean a Replacement mode is "repaired" in the same sense as other failure modes. Rather, it provides a way to specify the amount of time required to complete the replacement.

Read more: Modeling the Repair of Failure Modes (page 101).

Unlike the Preventive Maintenance mode, *a Replacement event automatically impacts all other failure modes for the component*. When the replacement occurs, all failure modes are replaced when the Replacement mode completes. If a failure mode is FMCV-based, the FMCV is set to zero (indicating that the component is new).

When a reliability element that is being modeled as a system undergoes replacement, all children automatically undergo replacement also.

Read more: Modeling a Reliability Element as a System with Child Elements (page 58).

Replacement triggers will often be defined as functions of locally available properties such as operating time or total time. For example, you may want to trigger a replacement every year of operating time. These variables are available as locally available properties within the trigger input field.

Read more: Common Reliability Element Locally Available Properties (page 62); Example: Modeling Component Maintenance and Replacement (page 158).

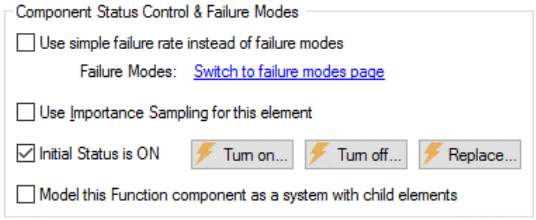
By default, like all failure modes, PM failure modes are added to the Internal Requirements list. As a result, when the PM is taking place, the element will be inoperable (and the Status for the element will report 1). However, you can choose to remove the PM failure mode from the Internal Requirements list. If you do so, when the PM is taking place, the element will be not be inoperable

(and the Status for the element will NOT report 1; if all other requirements are met, for example, it would report 0).

Read more: Common Reliability Element Outputs (page 60); Failure Modes and Internal Requirements (page 95).

Simulating Replacement Using the Replace Trigger

A replacement of a component can be directly triggered using the **Replace…** button in the main dialog for a reliability element. This resets the element and all its child elements back to their "as new" condition (i.e., it is identical to defining a PM:Replacement trigger).



Pressing the **Replace..** button will bring up the standard triggering dialog. Once a trigger has been specified, a checkbox in the bottom corner of the **Replace…** button will appear, and any triggers can be viewed by mousing over the button.

Within the dialog for triggering a Replacement, you can also specify Resource Requirements necessary to carry out the Replacement.

Note: The primary difference between using a Replace Trigger to simulate a replacement as opposed to using the Replacement failure mode is that the failure mode allows the repair to be assigned a duration. Replacements carried out using a Replace trigger are instantaneous.

Read more: Example: Modeling Component Maintenance and Replacement (page 158).

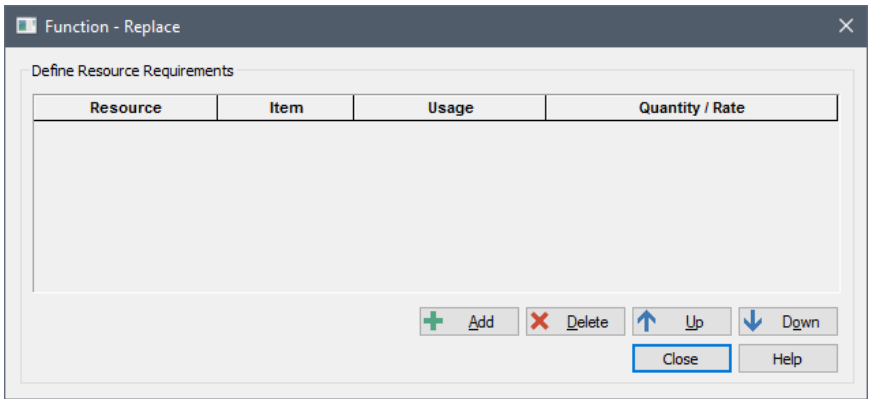
Specifying Resource Requirements for Replacement

A ***Resource*** is something that has a limited supply (e.g., spare parts, fuel, skilled personnel, money) and is required in order for elements of the modeled system to carry out certain actions.

Read more: Modeling Resources in the Reliability Module (page 121).

When simulating Replacement using the **Replace** trigger, you can specify that certain Resources must be available in order for the component to be replaced. If the Resources are not available when you try to replace the component, it will not be replaced until they are available.

To define Resource Requirement for Replacement, press the **Resources** button in the Replace Trigger dialog. The following dialog will be displayed:



Note: In order for this dialog to appear, you must have previously defined at least one Resource in the model.

You can add a Resource Requirement by pressing the **Add** button.

A Replace trigger interacts with the specified Resource Stores when it is triggered, and can only have one type of interaction (specified in the **Usage** column):

- **Spend (discrete):** A discrete quantity of the Resource is required in order to complete the Replacement. If the requested Resource quantity is not available, the trigger signal is "held", and it waits for the Resource to become available.

Modeling Resources in the Reliability Module

Sometimes desired actions can be constrained by Resource availability. For example, if a system component failed, its repair might not be possible until a mechanic and an appropriate spare part were available. In this case, the pool of mechanics and the stock of spare parts would be considered as Resources. Some resources, such as spare parts, are consumed when they are used. Others, such as mechanics, are only borrowed, and become available again once they are no longer required.

One of the advanced features in GoldSim is the ability to create and interact with Resources. Within GoldSim, a ***Resource*** is defined as something that has a limited supply (e.g., spare parts, fuel, skilled personnel, money) and is required in order for elements of the modelled system to carry out certain actions.

Read more: Example: Modeling Resource Requirements for Reliability Elements (page 170).

Using Resources involves three different steps:

- You must define a Resource Type (e.g., Gasoline, Electricians, Pump Motors) and specify its characteristics (discrete or continuous).
- You must specify where the ***Stores*** for that particular Resource exist in your model. Stores represent stockpiles or places where the actual Resource (e.g., parts, personnel) is stored or located when not being used. That is, Resource Stores can be thought of as having physical locations in the system you are modeling.

- You must specify how particular elements in the model interact with (consume, borrow from, or add to) the Resource Stores. Only certain kinds of elements can interact with Resources.

Note: The manner in which Resource Types are defined and Stores are created is described in detail in Chapter 10 of the **GoldSim User's Guide**.

Elements can interact with a Resource Store in three different ways:

- An element can *Spend* (consume*)* a Resource from the Store.
- An element can *Borrow* a Resource from the Store for a particular amount of time and then return it.
- An element can *Deposit* (generate*)* a Resource into a Store.

Only certain kinds of elements in GoldSim can interact with Resources. The table below summarizes how Reliability elements can use Resources:

Element	Resource Store Interaction	Allowed Usage	Comments
Functions and Actions	**Turn Component On**: Interacts when component is turned On via the On Trigger	Spend (discrete) Borrow (discrete) Deposit (discrete)	If Requirement is Spend or Borrow, and Resource is not available, the trigger is "held", and waits for the Resource to become available. Borrowed Resources are returned when the component is turned Off.
Functions and Actions	**Replace Component**: Interacts when component is Replaced via the Replace Trigger	Spend (discrete)	If Resource is not available, the trigger is "held", and waits for the Resource to become available.
Functions and Actions	**Repair Failure Mode**: Interacts when component is automatically repaired via the Resources button on Failure Modes dialog	Spend (discrete) Borrow (discrete)	If Resource is not available, the trigger is "held", and waits for the Resource to become available. Borrowed Resources are returned when the repair completes.
Actions	**Trigger an Action**: Interacts when the Action is triggered via the Action Trigger	Spend (discrete) Borrow (discrete) Deposit (discrete)	If Requirement is Spend or Borrow, and Resource is not available, the trigger is "held", and waits for the Resource to become available. Borrowed Resources are returned when the Action completes.

Element	Resource Store Interaction	Allowed Usage	Comments
Functions and Actions	**Operate the Component**: Interacts while component is operating via the Resources button on main dialog	Spend Rate Deposit Rate	While component is operating, spends or borrows at specified rate. If Spend Rate is specified and Resource is not available, the component stops operating. It resumes when enough Resource is available to cover the next timestep.

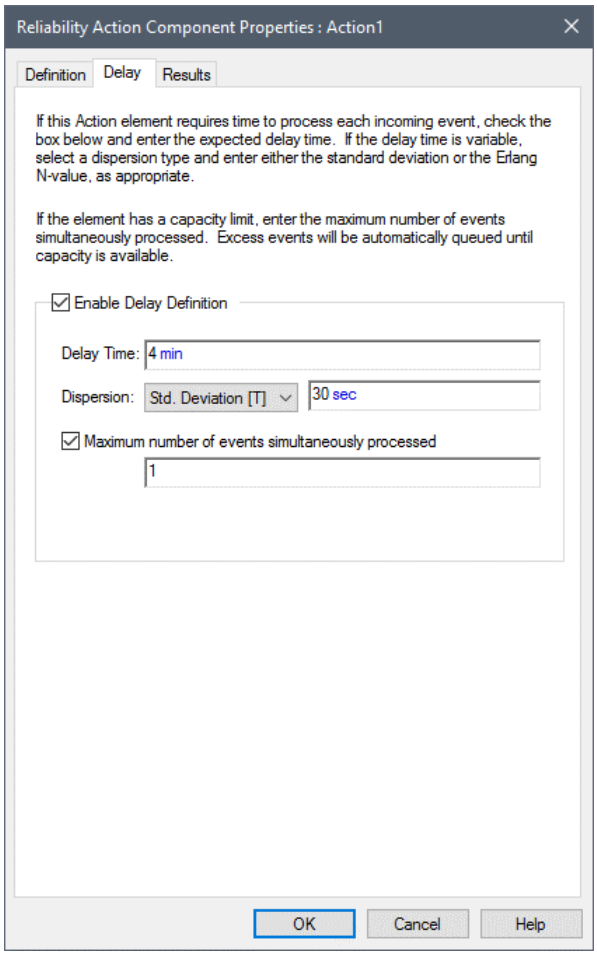
Read more: Specifying Resource Requirements for Turning Components On (page 91); Specifying Resource Requirements for Replacement (page 120); Specifying Resource Requirements for Repairs (page 104); Specifying Resource Requirements for Triggering an Action (page 75); Specifying Operating Resource Requirements (page 73).

Advanced Features of the Action Element

Two of the advanced features of Action elements, simulating delays and handling actions internally, are discussed below.

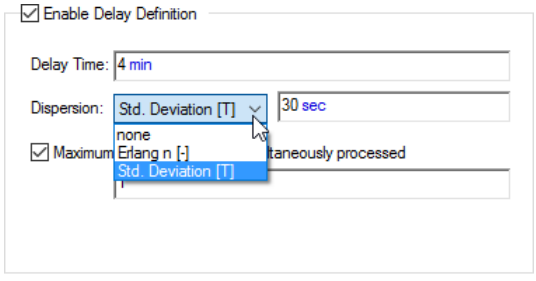
Adding Delays to Action Elements

In some cases, it may take time for an Action element to complete its action. GoldSim provides features that allow you to model this, and these are accessible in an additional **Delay** tab available in the Action element's property dialog:



By default, actions take place instantaneously. To enable delay features in an Action element, you must first check the **Enable Delay Definition** box. After doing so, you can specify a **Delay Time** (in any time units) using a number, expression or a link.

The **Dispersion** drop down box and input field allow you to add variability to the delay. The default is no dispersion. However, you can also specify two alternative ways to quantify the degree of dispersion in the signal:



If "Erlang n" is selected, you must enter a dimensionless value greater than or equal to 1. As n increases, the degree of dispersion decreases. As n goes to infinity, the dispersion goes to zero. The maximum amount of dispersion allowed is represented by n = 1.

If "Std. Deviation" is selected, you must enter a value with dimensions of time. The value must be greater than or equal to zero and less than or equal to the Delay Time. As the Std. Deviation decreases, the degree of dispersion decreases.

When the Std. Deviation goes to zero, the dispersion goes to zero. The maximum amount of dispersion allowed is represented by Std. Deviation = Delay Time.

The Erlang n and the Std. Deviation are related by the following equation:

$$n = \left(\frac{\text{Delay Time}}{\text{Std. Deviation}} \right)^2$$

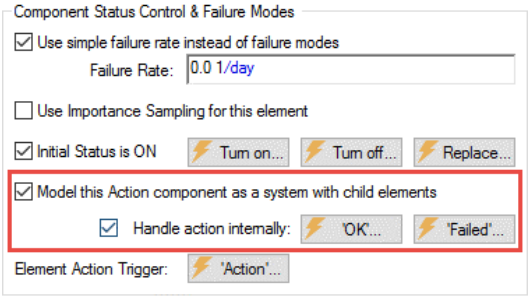
The number of actions that can be processed at any one time can be limited by selecting the **Maximum number of events simultaneously processed** checkbox. You can then use the input field to specify an integer number of events, either with a number, expression, or a link to another element.

If an Action element's action is triggered and the maximum number of events is already being processed, excess triggering events are queued until they can be processed by the element.

If an Action element stops Operating while an Action delay is underway, the delayed Action trigger's progress is frozen until the element is repaired, and then restarted after repair.

Handling Actions Internally

Within the Action element dialog, there is an option to **Handle action internally**:



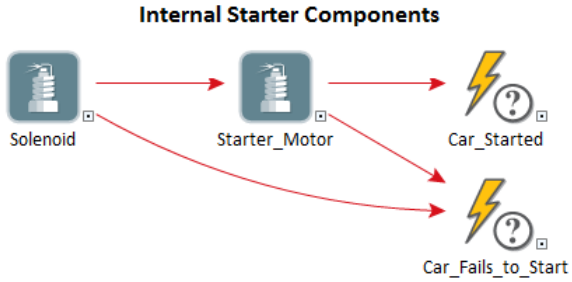
This option is only available if the Action is being modeled as a system.

Read more: Modeling a Reliability Element as a System with Child Elements (page 58).

Even if the Action is being modeled as a system, the checkbox is defaulted off.

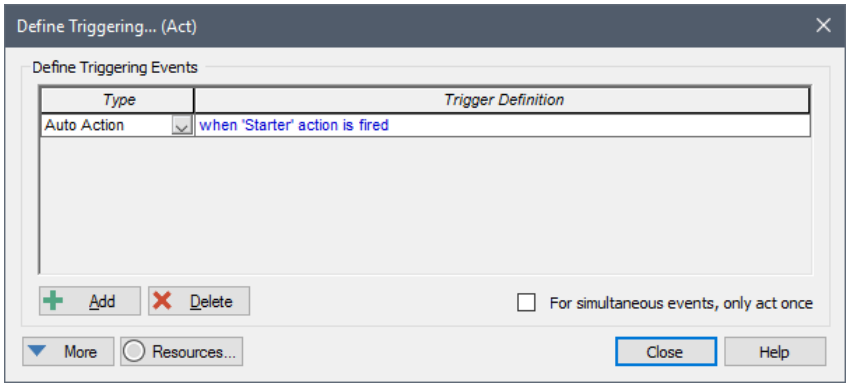
This checkbox controls how Actions that are defined as systems handle action triggers. If the box is checked, the Action triggers are automatically routed to internal Action components and the OK and Failed triggers become available (for handling responses from internal elements).

In order to handle an action internally, the triggering of the parent's action must automatically trigger a child action element or element(s). You can then implement as much logic as you desire (using any available GoldSim elements) to determine whether the internal (child) action is completed successfully or fails, but it must result in either a single event or condition that indicates success or a single event or condition that indicates failure. These "single events or conditions" must then be connected to the **OK** and **Failed** triggers inside the parent.



In this example, the Solenoid and Starter_Motor Actions are children of a Starter Action. The Solenoid is automatically triggered by the parent. It then triggers the Starter-Motor to act. If either the Solenoid fails or the Starter_Motor fails, the Car_Fails_to_Start event is triggered. If the Starter_Motor completes its action, the Car_Started event is triggered. The Car_Started event triggers the parent's **OK** trigger. The Car_Fails_to_Start event triggers the parent's **Failed** trigger.

To facilitate this, a special trigger type is available within the Action trigger when the Action element being triggered is the child of another action element, and the Parent element's **Handle action internally** option has been checked:

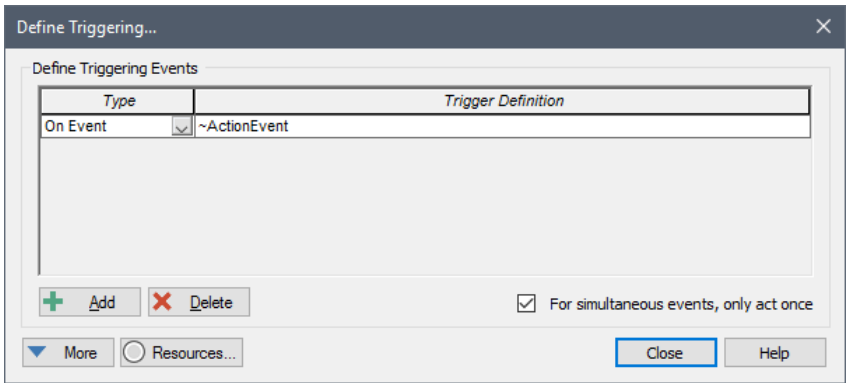


The Auto Action trigger automatically fires when the parent Action element is triggered. With this trigger type, the trigger expression cannot be edited.

In some cases, you may want the parent Action element to not only trigger internal child Action elements, but other kinds of elements inside the parent Action (e.g., a Discrete Change or a Triggered Event). To facilitate this, GoldSim provides a special locally available property that can be referenced inside Action elements that are defined to be handling events internally.

Note: Locally available properties derive their name from the fact that they may only be available in "local" parts of your model (e.g., within a particular element, or within a particular input field for an element). Locally available properties are discussed in detail in Chapter 10 of the **GoldSim User's Guide**.

In particular, an event called ActionEvent is available inside the parent Action, and this can be used to trigger elements inside the parent whenever the parent is triggered. It would be used as follows:



In this case, the child element inside the parent would automatically be triggered when the parent was triggered.

As pointed out above, when handling an action internally, internal calculations within the parent must result in either a single event or condition that indicates success or a single event or condition that indicates failure. These "single events or conditions" must then be connected to the **OK** and **Failed** triggers in the parent.

Note: The **OK** and **Failed** triggers of the parent can only be linked to outputs from child elements within it.

Warning: The parent element requires exactly one response from the children for each triggering event, as it cannot distinguish between multiple event triggers. Therefore, the modeler should not construct logic whereby one parent's action trigger could lead to the generation of multiple OK or multiple Failed events, as this could lead to unintended model behavior.

If the **'OK'** trigger is fired, and the parent element has no unrepaired failure modes, the parent element will successfully complete its action and emit an **ActionOK** event. However, if the **'OK'** trigger becomes true, and the parent has unrepaired failure modes, the parent element will emit an **ActionFailed** event.

If the **'Failed'** trigger is fired, the parent has not successfully completed its action, and an **ActionFailed** event will be emitted regardless of whether or not the parent element has unrepaired failure modes.

Like the **Action button**, the LED's in the bottom corner of the **'OK'** and **'Failed'** buttons will light once triggers are specified, and the triggers can be viewed by mousing over the two buttons.

When handling Actions internally, and both the parent and the child Actions have delays, the following sequence is used when the parent's Action is triggered:

- The child is triggered, and its Action delay begins.
- When the child's delay is completed, an ActionOK or Action Failed event is issued by the child, and this is transmitted to the **Handle action internally** triggers of the parent Action element

- The parent delay then begins, and when it completes, the parent issues an ActionOK or ActionFailed event.

Read more: Example: Handling Actions Internally (page 163).

Chapter 6: Displaying Reliability Results

> As a rule, software systems do not work well until they have been used, and have failed repeatedly, in real applications.
>
> Dave Parnas

Chapter Overview

This chapter describes the results and analysis tools available within the GoldSim Reliability Module.

In this Chapter

This chapter discusses:

- Saving and Accessing Reliability Results
- Availability and Reliability Results Summary
- Exporting Availability and Reliability Summary Results
- Availability and Reliability Histories and Statistics
- Failure Times Statistics
- Repair Times Statistics
- Causal Analysis
- Reliability Element Status and Failure Mode Histories and Statistics
- Defining and Using the Output Interface for a Reliability Element

Saving and Accessing Reliability Results

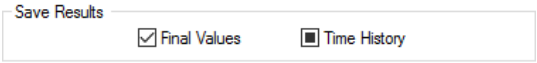
Function and Action elements have three different types of results that can be saved and viewed:

- Those that are actual outputs of the element itself (and can be referenced by other elements);
- Specialized analysis results; and
- Internal outputs (of systems) and local properties that are exposed through the Reliability element's Output Interface.

Function elements have five outputs that falls into the first category: the primary output of the element (which represents the component's status), the Failed output (which represents the status of the element's failure modes), and three reliability and availability outputs. The Action element has an additional two outputs of this type: a count of the number of successful and unsuccessful actions.

Read more: Common Reliability Element Outputs (page 60); Outputs Available Only for Action Elements (page 77).

You specify whether or not results for these outputs are saved by clicking the **Final Values** and/or **Time History** checkboxes at the bottom of the **Definition** tab for the element:



Like all element outputs, these outputs can be accessed by right-clicking on the output port of the element.

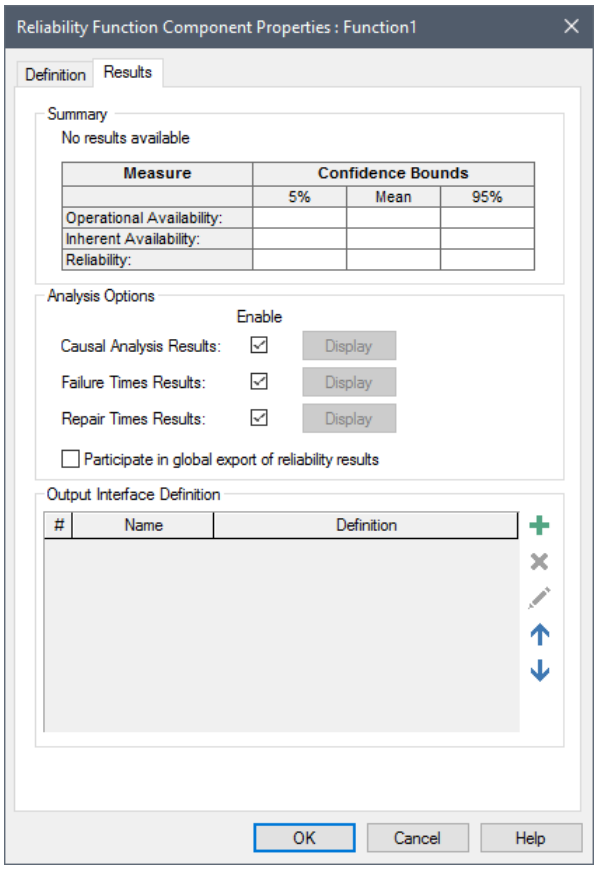
Note: The **Final Values** checkbox always controls whether Final Value results are saved. However, the **Time History** checkbox can be overridden. In particular, it is always applied for single realization runs, but is overridden for multiple realization runs. In particular, Time History results for multiple realization runs are only saved for outputs that are connected to Result elements.

Note: By default, the **Failed** output is not set to save time history results even for single realization runs (since it is an array). To save time history results for the **Failed** output, you must connect it to a Time History Result element or right-click on it in the output interface and select the Save Time History option.

Read more: Availability and Reliability Histories and Statistics (page 134).

Note: Basic concepts associated with saving and viewing results in GoldSim are discussed in Chapter 3 of the **GoldSim User's Guide**.

In addition, GoldSim provides a number of reliability analysis results on the **Results** tab of Function and Action elements:



The "Summary" portion of the dialog automatically displays the key reliability metrics after a simulation is run.

Read more: Availability and Reliability Results Summary (page 132).

The "Analysis Options" portion of the dialog provides three additional types of analyses (Causal Analysis, Failure Times and Repair Times). GoldSim will *not* carry out these analyses after a simulation is run unless the boxes are checked. By default, these are checked on. If a particular analysis has been saved, then after running the simulation the results can be viewed by returning to this tab and clicking on the **Display** button.

Read more: Failure Times Statistics (page 136); Repair Times Statistics (page 137); Causal Analysis (page 138).

Note: The **Display** buttons are grayed out when the model is in Scenario Mode. That is, you cannot view these results when running and comparing scenarios.

The **Participate in global export of reliability results** checkbox allows you to indicate whether the element's Summary statistics should be exported (to a spreadsheet) during a global reliability results export.

Read more: Exporting Availability and Reliability Summary Results (page 132).

The "Output Interface Definition" portion of the dialog allows you to specify customized outputs associated with the element (e.g., if the element was system, this might by an output that exists inside the element).

Note: For any outputs created in the Output Interface Definition, whether or not results for these outputs are saved is determined by the **Final Values** and/or **Time Histories** checkboxes at the bottom of the **Definition** tab for the element:

Read more: Defining and Using the Output Interface for a Reliability Element (page 142).

Availability and Reliability Results Summary

The "Summary" portion of the dialog automatically displays the key reliability metrics (in terms of a mean and 5%/95% confidence bounds) after a simulation is run:

Summary

Results for 1000 realizations with mean duration of 3 hr.

Measure	Confidence Bounds		
	5%	Mean	95%
Operational Availability:	0.8404	0.8451	0.8498
Inherent Availability:	0.9237	0.9284	0.9331
Reliability:	0.3429	0.3680	0.3931

Mean Operational Availability. This statistic is the average fraction of time the component has been operating over the simulated time period. For a steady-state system, it represents the probability that the component will be operating at any given time.

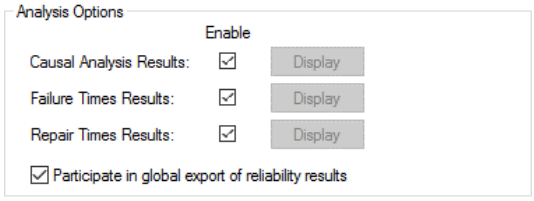
Mean Inherent Availability. This statistic is the average fraction of time the component has been operable over the simulated time period. For a steady-state system, it represents the probability that the component will be operable at any given time. A component that is turned off or has unmet external requirements, but has not failed would be considered operable. Hence, the inherent availability is always greater than or equal to the operational availability.

Reliability. This statistic is the fraction of realizations in which the component survived for the entire simulated time period. Hence, it represents the probability that the component will be operable for the entire simulation. For a steady-state system (with repeated failures and repairs) the system, by definition, has a 0 reliability.

Exporting Availability and Reliability Summary Results

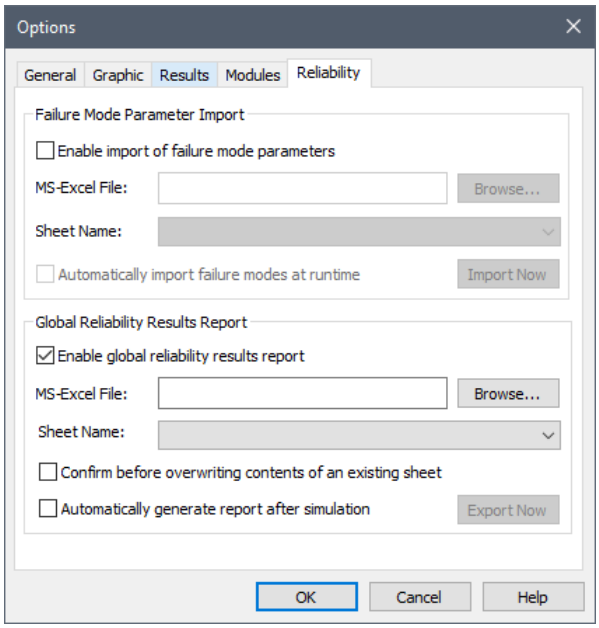
GoldSim allows you to export a report with summary statistics (Operational Availability, Inherent Availability and Reliability) to Microsoft Excel from Reliability elements.

You can select which elements are to be exported (i.e., Summary statistics do not need to be exported for all Reliability elements). To indicate that you wish to export the Summary statistics for a particular Reliability element, you simply check the **Participate in global export of reliability results** option in the "Analysis Options" portion of the **Results** tab for the element:



Note: Cloning and versioning are not supported for the **Participate in global export of reliability results** option.

To enable the global export, select **Model|Options** from the main menu, click on the **Reliability** tab of the dialog, and check the **Enable global reliability results export** option (in the bottom half of the dialog):



You must then specify an Excel spreadsheet filename and the sheet where failure mode data is to be exported. The spreadsheet does not have to exist (if it does not exist, GoldSim will create it). For existing spreadsheet files, the **Sheet Name** drop down will show all sheets in the specified worksheet along with an "Append New Sheet (Date and Time as sheet)" option. If this is selected, GoldSim will create a new sheet, with the Date and Time as the sheet name.

If a spreadsheet filename is specified but does not yet exist, the "Append New Sheet (Date and Time as sheet)" option is the only option.

The **Confirm before overwriting the contents of an existing sheet** option prompts the user if data in an existing sheet will be overwritten by exported data. The **Automatically generate report after simulation** option will automatically export results at the conclusion of the simulation. If this option is cleared, data can be exported manually in Result Mode using the **Export Now** button.

When results are exported, the format is as follows:

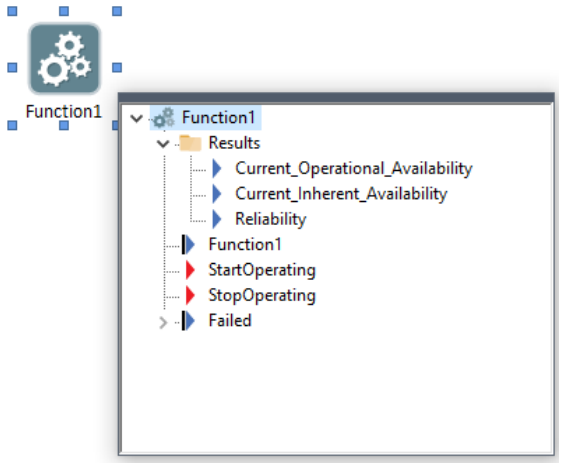
- The first 6 rows provide run information (e.g. GoldSim version, filename, etc.)

- Rows 7 through 9 are headers
- Column A: Element path
- Column B: Element ID
- Column C: Operational Availability (5%)
- Column D: Operational Availability (Mean)
- Column E: Operational Availability (95%)
- Column F: Inherent Availability (5%)
- Column G: Inherent Availability (Mean)
- Column H: Inherent Availability (95%)
- Column I: Reliability (5%)
- Column J: Reliability (Mean)
- Column K: Reliability (95%)
- Column L: Mean Time to Failure
- Column M: Mean Time to Repair

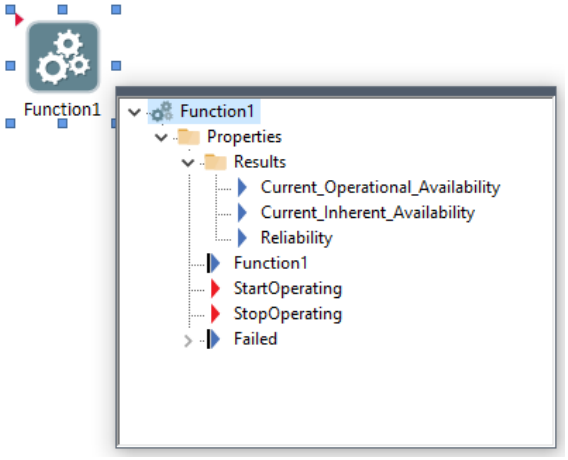
Note: GoldSim exports the results sorted alphabetically first by path, and then by Element ID.

Availability and Reliability Histories and Statistics

Within the output port for a reliability element there are three outputs provided in a folder named Results:



Note that if the element is being modeled as a system with child elements, these outputs are contained within a "Properties" folder when viewing the output port for the element:



Current_Operational Availability: This is the fraction of time the element was *operating* over a specified time period immediately previous to the current time. The specified time period is approximately 2% of the simulation duration.

Current_Inherent Availability: This is the fraction of time the element was *operable* over a specified time period immediately previous to the current time. A component that is turned off or has unmet external requirements, but has not failed would be considered operable. Hence, the Inherent Availability is always greater than or equal to the Operational Availability. The specified time period is approximately 2% of the simulation duration.

Reliability: This value is 1 if the element has not failed, and 0 if it has failed. Unlike the availability results, no averaging period is required. At any point in time, the element has either failed or has not failed.

Note: These two availability results represent current (or, conceptually, instantaneous) values. They are different from and should not be confused with the availability results that are presented on the Results tab of Reliability elements, which represent the availability over the entire duration of the simulation.

Read more: Availability and Reliability Results Summary (page 132).

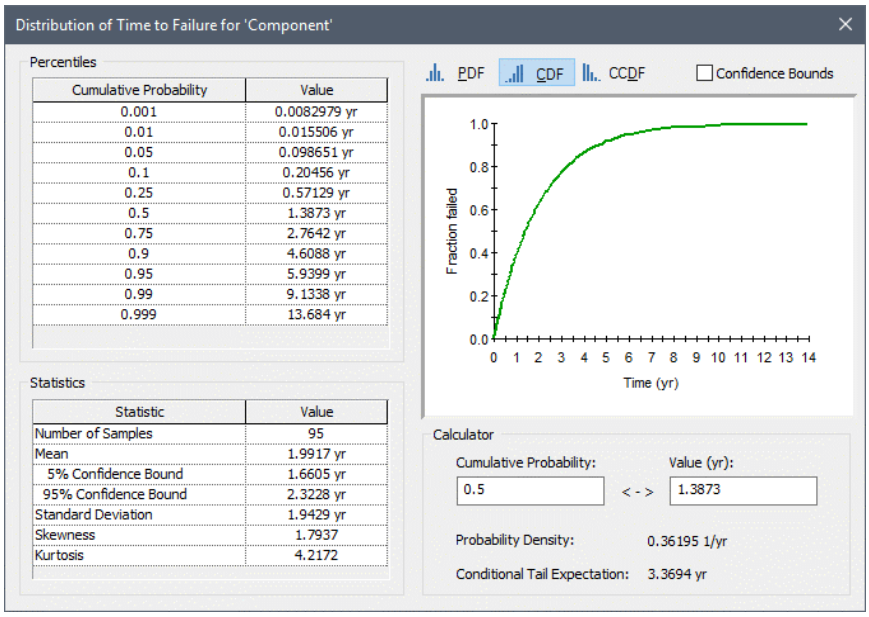
When viewing the current availability results, you must interpret them with caution. This is because they represent the "instantaneous" value of the statistic at any given point in time (as opposed to the cumulative value of the statistic). It is computed by dividing time into "slices" or "windows" for a time period just previous to the plot point (with a length of approximately 2% of the duration). Therefore, the plot is most useful if a large number of Monte Carlo realizations have been run, and you are viewing probability histories. Viewing single realizations is of limited value.

Similarly, if you were to plot the distribution of one of the availability outputs and note the mean, you would see that the result would most likely be different from the result shown in the Summary section of the Reliability element's Result tab. This is because the distribution results would be based on an instantaneous metric (the availability for the last 2% of the simulation duration), while the Summary results are based on a cumulative metric (the availability over the entire simulation).

These considerations do not apply for the Reliability output, as no averaging period is required: at any point in time, the element has either failed or has not failed.

Failure Times Statistics

The **Failure Times** button in the **Results** tab of a reliability element allows you to view a distribution of the time to failure (from all modes) for the component:



Note: This dialog is nearly identical to the Distribution Summary for a standard Distribution result. Viewing distribution results such as this is discussed in detail in Chapter 8 of the **GoldSim User's Guide**.

Note: The **Display** button for Failure Times is grayed out when the model is in Scenario Mode. That is, you cannot view these results when running and comparing scenarios.

The Mean value that is displayed represents the Mean Time to Failure (MTTF) for the component.

The times to failure represent the time intervals between when the component starts operating to when it fails (i.e., the time to failure). Of course, if repairs are taking place, a single realization of the system can have multiple time intervals between failures. The distribution that is displayed incorporates all intervals for all realizations.

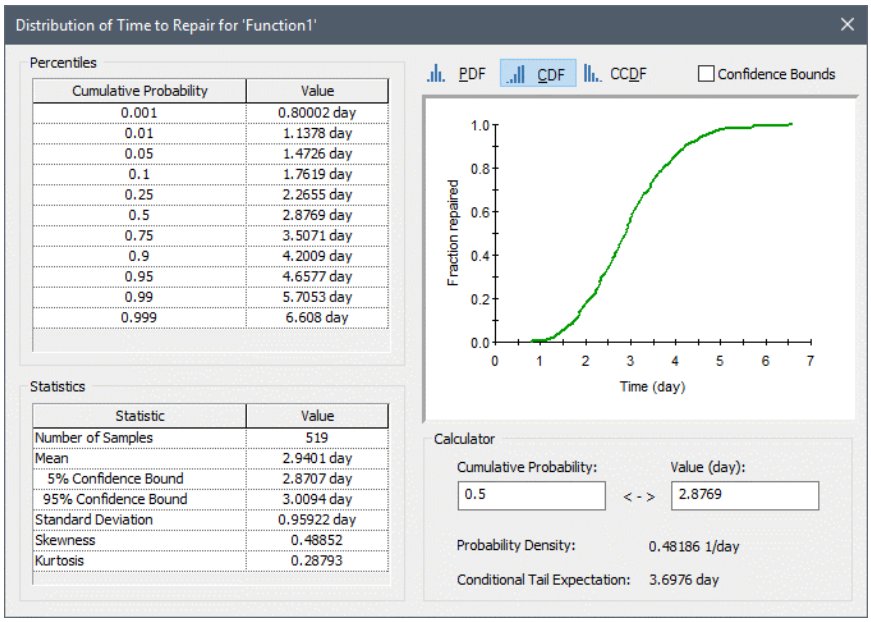
Note: If a component is being modeled as a system, a "failure" includes instances when the system fails directly and when it stops operating due to an internal requirement not being met. However, if it stops operating due to an external requirement not being met, this is not treated as a failure.

Ideally, a simulation will have produced a large number of failures and a well-defined distribution will have been developed by GoldSim. However, in some cases, the component will be unfailed at the end of the realization. This represents a censored sample of the failure time of the form "Failure time > X" , where X represents the time that the element had been operating as of the end of the realization. The manner in which this information is used to compute the failure distribution is described in Appendix A.

Note: GoldSim will only plot a failure distribution if there are at least 10 failure records (a failure record being either an actual failure time or a failure time of the form "Failure time > X"). In addition, at least 25% of the records must be an actual failure time (rather than a censored sample).

Repair Times Statistics

The **Repair Times** button in the **Results** tab of a reliability element allows you to view a distribution of the time to repair for the component:



Note: This dialog is nearly identical to the Distribution Summary for a standard Distribution result. Viewing distribution results such as this is discussed in detail in Chapter 8 of the **GoldSim User's Guide**.

Note: The **Display** button for Repair Times is grayed out when the model is in Scenario Mode. That is, you cannot view these results when running and comparing scenarios.

The Mean that is displayed represents the Mean Time to Repair (MTTR) for the component.

The times to repair represent the time intervals between when the component fails and when it is repaired and starts operating again. Of course, a single realization of the system can have multiple repairs. The distribution that is displayed incorporates all repairs for all realizations.

Causal Analysis

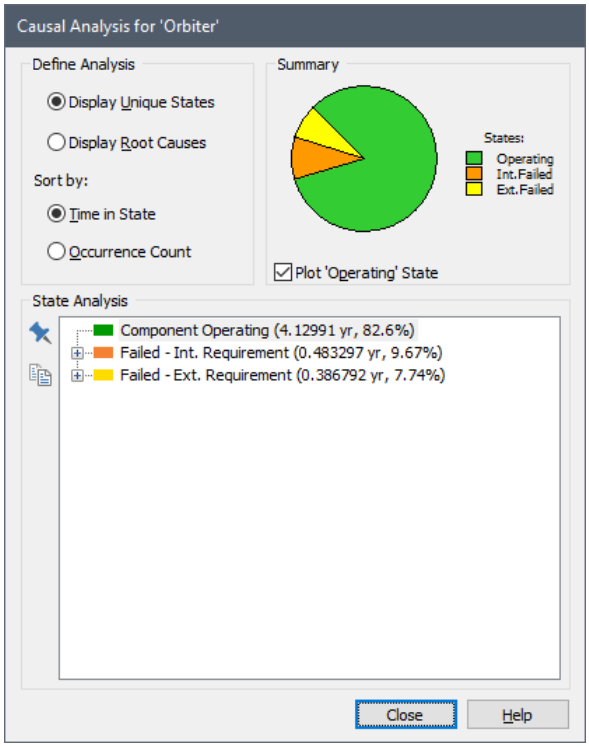
GoldSim records the unique state of the element each time that it fails. After the simulation has been completed, GoldSim can analyze these failure scenarios to allow you to determine key sources of unreliability for a particular component.

The **Causal Analysis** option in the **Results** tab of a reliability element allows you to view this type of analysis for the component.

Note: Causal Analysis is not available if you have carried out your simulation using the Distributed Processing Module.

Note: The **Display** button for Causal Analysis is grayed out when the model is in Scenario Mode. That is, you cannot view these results when running and comparing scenarios.

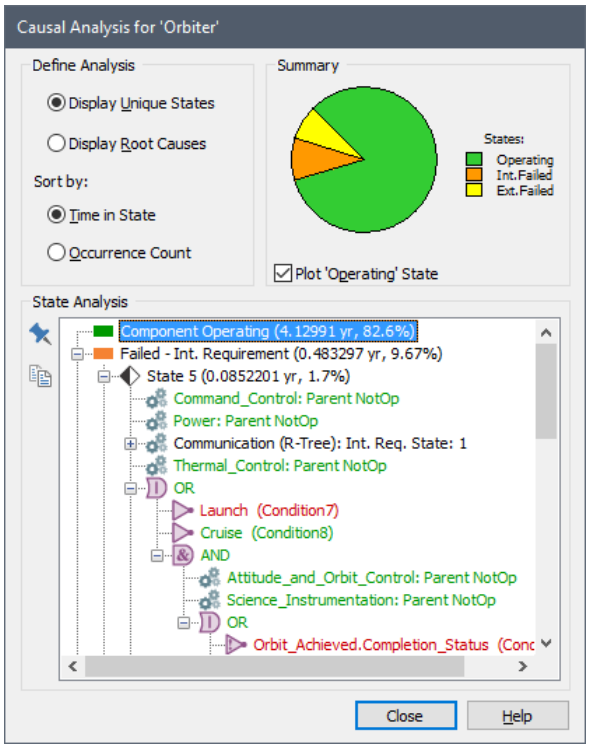
When you first click the **Display** button for **Causal Analysis**, a dialog will appear with the component's unique states analyzed and sorted by **Time in State**:



The Summary section of the dialog displays a pie chart which shows the average percentage of the total simulation time spent in each of a number of ***primary states*** (Operating, Failed due to Internal Requirements, Failed due to External Requirements, Preventive Maintenance, Parent Not Operating and Component

Turned Off). The Operating state can be excluded from the pie chart by clearing the "Plot 'Operating' State" checkbox (so that only non-operating states are shown in the pie chart).

Below the summary, the "State Analysis" section sorts each of the primary states by the time in each state (normalized to a single realization). For the External Requirements and Internal Requirements, the tree can be expanded (by clicking the plus sign next to the entry for those primary states) to explore each of the unique substates the component experienced:



Each of the substates for each primary state is again sorted by time in state. To allow you to quickly identify key sources of unreliability, for Failed Internal Requirements and Failed External Requirements primary states, the tree nodes for each individual substate are expanded in the section and color-coded: nodes that are true are highlighted in green and nodes that are false are highlighted in red.

Note: You can jump to the next false node in the tree by pressing **Ctrl+N**, and you can jump to the previous false node by pressing **Ctrl+P**. All branches of the state analysis tree can be expanded by pressing **Ctrl+A**.

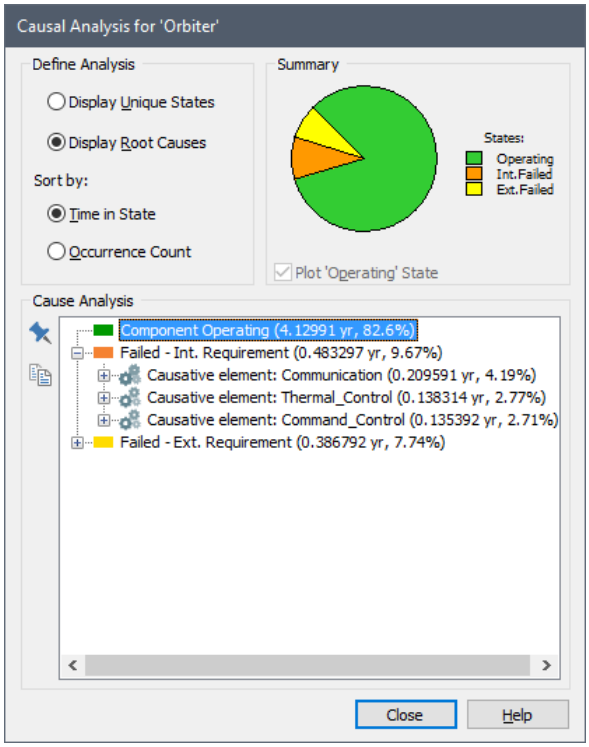
Alternatively, each of the primary states and their substates can also be sorted by **Occurrence Count**. When this radio button is selected, GoldSim tracks the number of times that the element entered each primary state, as well as the number of times that GoldSim entered each substate for the Failure Mode, Failed Internal Requirements and Failed External Requirements. The primary states, and their substates are then sorted using these values. Instead of displaying the time in each state (normalized to a single realization), each primary and substate displays the state frequency (count) normalized to a single realization.

GoldSim can also display the unique state information in a different manner, by root cause. If you select the **Display Root Cause** radio button, GoldSim displays the primary states in the same manner, but the substates are analyzed and GoldSim attempts to determine the identity of the lowest level components that resulted in failure of the system.

Note: The algorithm that is used to carry out the Root Cause analysis is described in Appendix A.

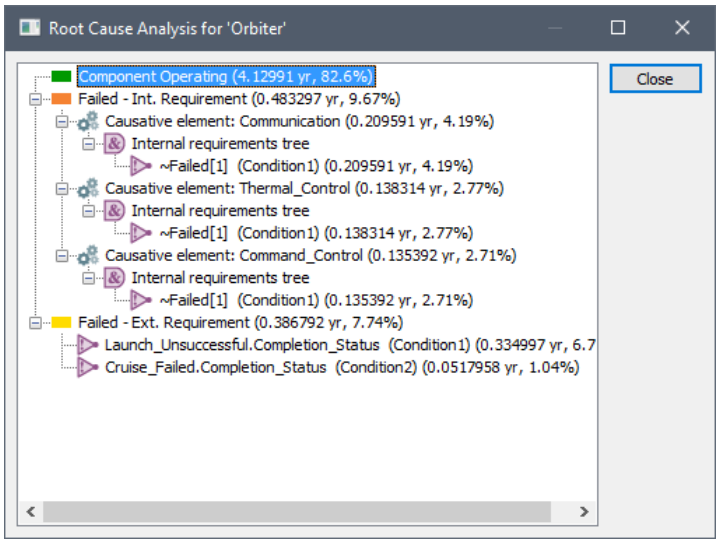
These states are then sorted by cause (e.g., a failure mode, or the failure of a sub-component) most frequently responsible for failure of the component. Depending on whether Time in State or Occurrence Count is selected, for each state, GoldSim displays either the time or count.

Root causes can also be expanded to show the reason why reliability components identified as root causes were not operating or operable. These reasons can include conditions in the element's internal or external requirements tree, failure modes of the component, or the fact that the component was turned off:



Pushpin button

Note that if you have a large causal analysis tree with a number of hierarchical levels, it might be useful to view the tree in an expanded sub-window. You can do this by pressing the "pushpin" button in the Result tab just to the left of the tree. When you do so, GoldSim will display the tree in a separate (and resizable) window:



Reliability Element Status and Failure Mode Histories and Statistics

In addition to the analysis and results provided on the Results tab for a reliability element, in some cases it may be of value to view the time history and probability distribution of the status of the element.

The status output (the primary output for the element) is an integer that indicates the Status of the component. At any given time, it takes one of the following values:

Output Value	Component Status
0	All requirements are met, the component is not failed, it is turned on and operating.
1	A preventive maintenance (that makes the component inoperable) is underway.
2	Internal requirements are not met.
3	External requirements are not met.
4	Element is not turned on.
5	Parent element is not operating.
6	An operating Resource requirement is not met.

Note: By default, preventive maintenance is defined as an Internal Requirement (i.e., when active, the component cannot operate). However, optionally, you can specify that preventive maintenance can be carried out while the component continues to operate.

If more than one status value applies at any given time, then the smallest value is output.

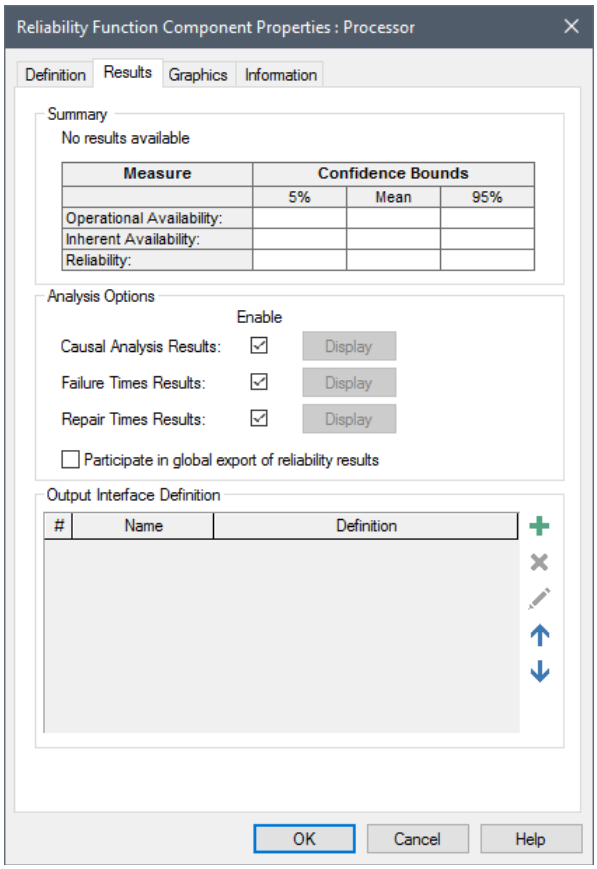
Plotting a time history of a component's status can provide insight into how the component behaves through time. A distribution result of the status at any particular point in time (e.g., end of realization) could also be useful in some circumstances. This distribution, of course, will be discrete (rather than continuous).

Read more: Example: Using the Reliability Element's Primary Output (page 146).

Defining and Using the Output Interface for a Reliability Element

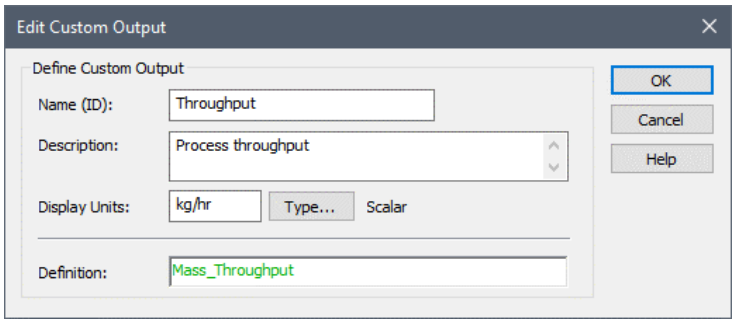
In some cases, to make your model logic more transparent, you may want to define your own "custom" outputs for a Reliability element. These outputs can be functions of local properties of the Reliability element itself (e.g., it's Status), or, if the element is being modeled as a system, they may be outputs (or functions of outputs) of elements *inside* the Reliability element (e.g., a throughput rate if the element represented some processing step).

To facilitate this, GoldSim allows you to create these custom outputs via an Output Interface Definition section on bottom half of the **Results** tab of Reliability elements:



Add button

In order to add a custom output, press the Add button in the Output Interface Definition portion of the dialog. When you do this, GoldSim displays the following dialog:



The **Name (ID)** is the name by which the output can be referenced in the model. For example, if the name of the Reliability element was X, and the name of the output was Y, the output could be referenced as X.Y. This ID has the same restrictions as an element ID.

The **Description** is an optional description of the output. The **Display Units** represent the units in which you wish the output to be displayed. The **Type...** button is used to access a dialog for specifying the data type (e.g., value/condition, scalar/vector/matrix).

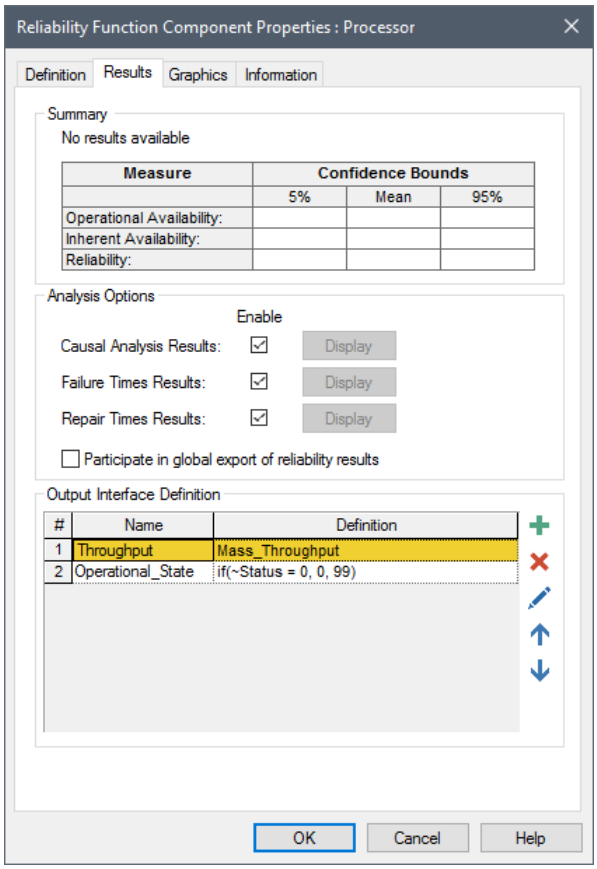
You then must enter the definition of the output in the **Definition** field. This must be consistent with the specified display units and type. This is a standard input field, and as such, can be a link or an expression. In most cases, the Definition will reference either locally available properties of the Reliability element (e.g., the Status) or one or more outputs from inside of the element itself (if it is being modeled as a system).

Read more: Common Reliability Element Locally Available Properties (page 62).

Note: In some cases, it may be necessary to reference outputs that are not associated with the Reliability element itself when defining a custom output. As a result, the Definition field can actually reference any accessible output in the model.

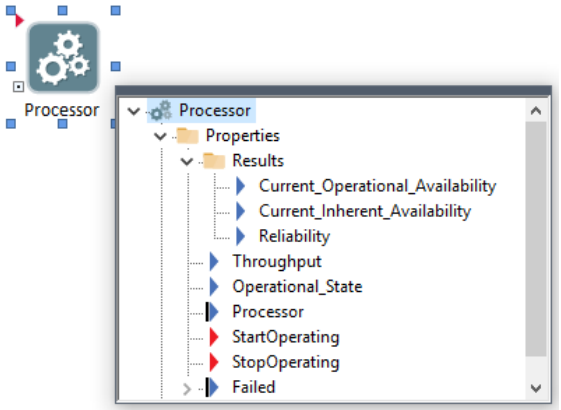
There is no limit to the number of custom outputs that can be added.

After adding the custom outputs, the Results tab displays them as follows (in this example, two custom outputs have been added):



Note that buttons for deleting, editing and moving the outputs up and down in the list are available directly below the Add button.

These outputs are then available on the output port for the element along with all of the other outputs:



Read more: Common Reliability Element Outputs (page 60).

Hence, in this example, they could subsequently be referenced elsewhere in the model as “Processor.Throughput” and “Processor.Operational_State”.

Read more: Example: Using Custom Reliability Outputs to Report Throughput Calculations (page 165).

Chapter 7: Example Reliability Module Applications

> Success is the ability to go from one failure to another with no loss of enthusiasm.
>
> Winston Churchill

Chapter Overview

This chapter describes a number of example models that are installed with the Reliability Module.

Note: All of the models described in this chapter can be found in the Reliability Examples folder in your GoldSim directory (accessed by selecting **File | Open Example...** from the main menu).

Some of the example models are designed to answer questions you might have about various concepts in the reliability module, while others show you how to accomplish more advanced modeling tasks.

In this Chapter

This chapter discusses:

- Example Models Illustrating Basic Reliability Module Concepts
- Example Models Illustrating Advanced Reliability Module Concepts
- Example Models Illustrating the Use of Basic GoldSim Elements with the Reliability Module

Example Models Illustrating Basic Reliability Module Concepts

The example models described below illustrate a number of basic concepts associated with the Reliability Module.

Example: Using the Reliability Element's Primary Output

Function and Action elements have a primary output with the same name as the element itself that displays the current state of the element. At any given time, it takes one of the following values:

Output Value	Component Status
0	All requirements are met, the component is not failed, it is turned on and operating.
1	A preventive maintenance (that makes the component inoperable) is underway.
2	Internal requirements are not met.
3	External requirements are not met.
4	Element is not turned on.
5	Parent element is not operating.
6	An operating Resource requirement is not met.

Note: By default, preventive maintenance is defined as an Internal Requirement (i.e., when active, the component cannot operate). However, optionally, you can specify that preventive maintenance can be carried out while the component continues to operate.

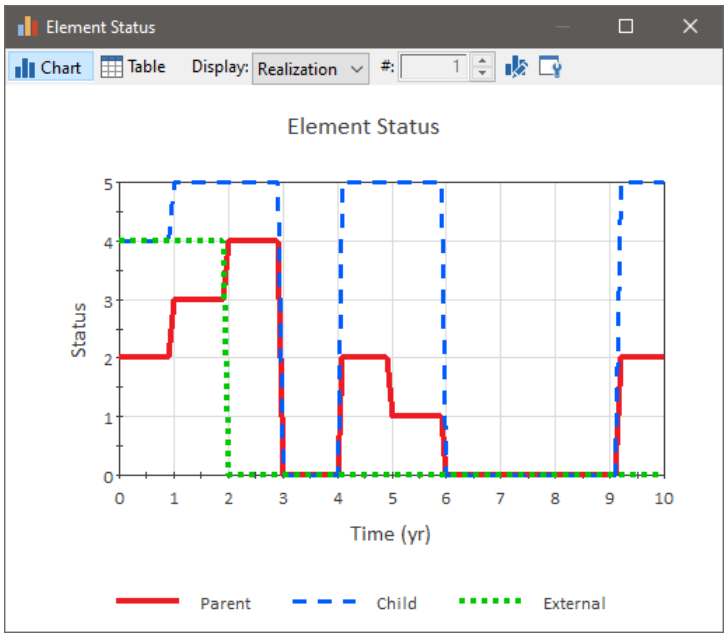
If more than one status value applies at any given time, then the smallest value is output.

Read more: Common Reliability Element Outputs (page 60).

The model file Status.gsm, found in the Reliability Examples folder in your GoldSim directory directory (accessed by selecting **File | Open Example...** from the main menu), shows situations that can lead to the different element status values. There are three elements: a Parent element (which is modeled as a system), a Child element, and an External element. The Parent element has an external requirement node that references the External element, and an internal requirement node that references the Internal element.

Read more: Defining Operating Requirements for Reliability Elements Using Logic Trees (page 64); Modeling a Reliability Element as a System with Child Elements (page 58).

The status outputs of the three elements are shown below:



At the start of the simulation, the External, Parent and Child elements are off. This means that the status of the External element is 4. However, the Parent and Child element are more complex because they are actually experiencing multiple states. The Parent element is initially off (4), has unmet external requirements (3), and unmet internal requirements (2). The Parent element outputs the value of the lowest numbered state it is experiencing (in this case, 2). The Child element is off (4) and its parent is also off (5), so it displays a status of 4.

After one year has passed, the Child element is turned on. Its status changes to 5 (as it is only experiencing the parent off state), and the status of the Parent changes to 3 (it is now off and has unmet external requirements).

Read more: Turning Components On and Off in the Reliability Module (page 90).

At 2 years, the External element is turned on. Its status changes to 0 (operating), and the Parent's status changes to 4 (off). The Child element's status is unchanged, as the parent is still off.

At 3 years, the Parent element is turned on. Its status changes to 0 (as it is now able to operate). The Child's status now changes to 0, as the Parent element is now operating.

At 4 years, the Parent experiences a failure and its status changes to 2 (since the failure was specified as being fatal to the component). The External element is unaffected, but the Child element changes to 5 (parent element not operating).

At 5 years, the Parent begins a 1 year preventive maintenance event. The Parent's status switches to 1, and the Child element's status is unchanged. The External element is unaffected. When the repair is completed (at 6 years), the statuses of the Parent and the Child elements change to 0.

In addition to using the Status to better understand what is actually happening in a particular realization, the Status output is often used to interface with other GoldSim elements. For example, an engine might withdraw fuel from a Reservoir while it is operating (i.e., the model would monitor the Status and

only do this of the Status was 0), or a generator might provide a certain amount of energy when it's operating.

Example: Modeling Dependencies on Other Reliability Components

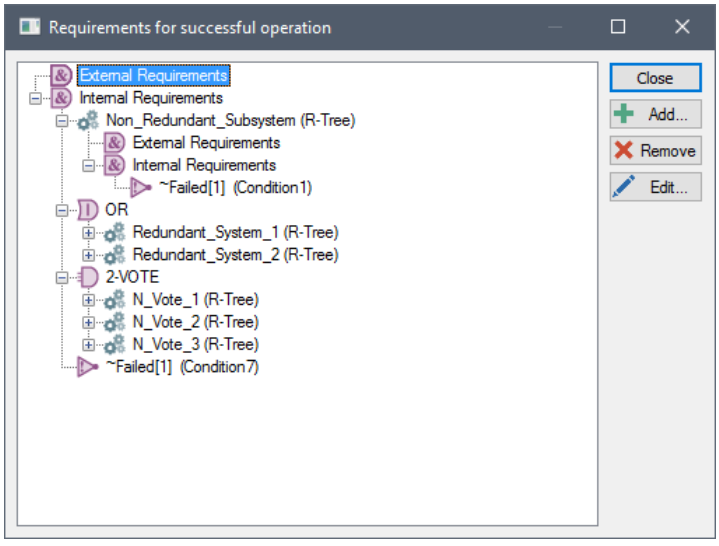
In many cases, successful operation of a particular reliability component is dependent on other reliability components being operable or operating (e.g., a flashlight requires a functional battery and a functional bulb to operate). In GoldSim, these types of dependencies are modeled using logic trees. You can select a requirements-tree (evaluates to true when the component's requirements are met), or a fault-tree (evaluates to false when the component's requirements are met).

Read more: Defining Operating Requirements for Reliability Elements Using Logic Trees (page 64); Modeling a Reliability Element as a System with Child Elements (page 58).

The top node in a requirements tree is always an External Requirements AND gate, used to reference peer elements. The Internal Requirements AND gate is used for referencing failure modes and for referencing child elements of Reliability elements that are being modeled as a system. For fault trees, the AND gates are replaced with OR gates.

GoldSim provides a number of nodes for building branches of the logic tree (AND, OR, and N-Vote gate nodes), as well as variable nodes for referencing conditions (Condition, which is true when the condition is true, and Not, which is true when the condition is false). However, the most frequently used nodes are the RL Component variable node (true when external components are operating, and true when child elements are operable), and the Not RL component variable node (true when external components are not operating, and true when internal components are not operable).

The model file Redundancy.gsm, found in the Reliability Examples folder in your GoldSim directory directory (accessed by selecting **File | Open Example...** from the main menu), illustrates modeling of dependencies between reliability elements. The model contains a Parent element, and a number of child elements the parent requires in order to operate. The Parent has one child that is not redundant, a group of two children where at least one child must be operable for the Parent to operate, and group of three children where two of the three children must be operable in order for the Parent to operate. Each child reliability element uses a simple failure mode with a failure rate of 2/5yr. The Parent element has no failures of its own. It's Requirements tree looks like this:



As you can see, all of the connections are made to the Internal Requirements AND node (as all of the subsystems are located inside the Parent_Element). Notice that the non-redundant system is directly connected to the AND node, while the redundant subsystem, and the two out of three subsystem are linked through an OR node and an N-Vote node respectively.

While this model shows the process for subsystems, the mechanism for referencing non-redundant systems, redundant systems and N-vote systems is the same for peer elements. The connections are simply made to the "External Requirements" AND node instead of the "Internal Requirements" node. If you would like to view the fault-tree equivalent of the requirements-tree of this model, simply open the Parent's dialog and select fault-tree from the "Logic tree represents a" drop-down list. This will convert the requirements-tree into an equivalent fault-tree.

Example: Understanding the Differences Between Failure Mode Base Variables

Most failure modes are defined relative to a failure mode control variable (FMCV). This is the variable that is referenced by the failure mode to determine when failure occurs. (For those failure modes that are defined as distributions, the control variable represents the x-axis of a failure distribution plot.)

For FMCV-based failure modes, the failure mode calculates the FMCV "age" that will result in its next failure at the start of the realization, after each time the mode is repaired, and after each time the component is replaced. When the FMCV exceeds this value, the failure occurs.

Read more: Failure Mode Control Variables (page 108).

Each (FMCV-based) failure mode has its own FMCV, which is defined in terms of a specified "base variable". Two built-in based variables are available to both the Function and the Action element. These are "Total time" which accrues time regardless of whether or not the component is operating and "Operating time", which accrues time when the component is operating. The latter is the default base variable for all failure modes in GoldSim. A third base variable, "Number of actions completed", is available only to the Action element.

Read more: Understanding Failure Mode Base Variables (page 109).

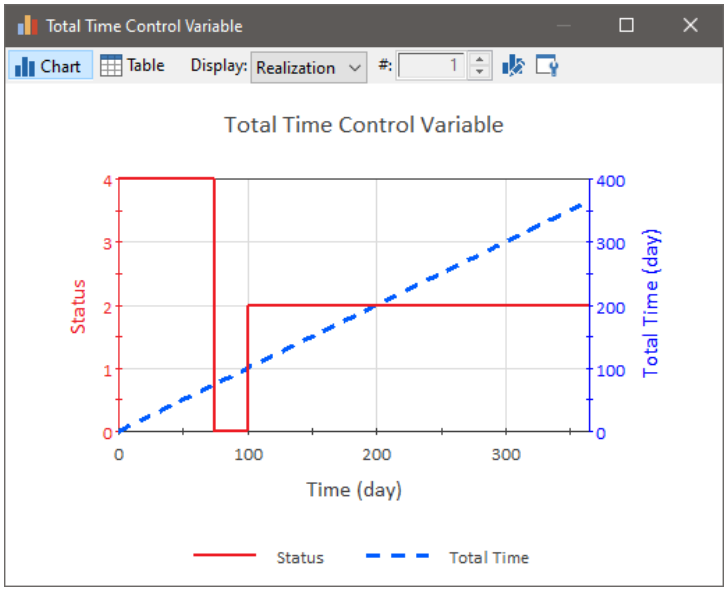
Base variables can be changed by selecting a Failure Mode on the **Failure Modes** tab of a reliability element and clicking the **Settings…** button. The failure mode base variable is selected from a drop-down list at the top of the dialog.

The two built-in base variables that reference time are quite different in the way they accrue time. The "Operating time" base variable accrues time only when the component is operating, whereas the "Total time" base variable accrues time even if the component or its parent has been turned off. Both can be reset when the failure mode is repaired or the component is replaced.

The model file ControlVariables.gsm, found in the Reliability Examples folder in your GoldSim directory directory (accessed by selecting **File | Open Example...** from the main menu), illustrates the difference between these three base variables. The model contains three elements, each of which has a single Specified Value Exceeded failure mode that fails at a value of 100 days (100 completed actions for the element using the Number of actions completed failure mode). All three elements are turned on at 75 days and turned off at 250 days. The Action element's action is triggered once per day, and the component that uses the Operating time control variable undergoes a Preventive maintenance event at 125 days that lasts for exactly 5 days.

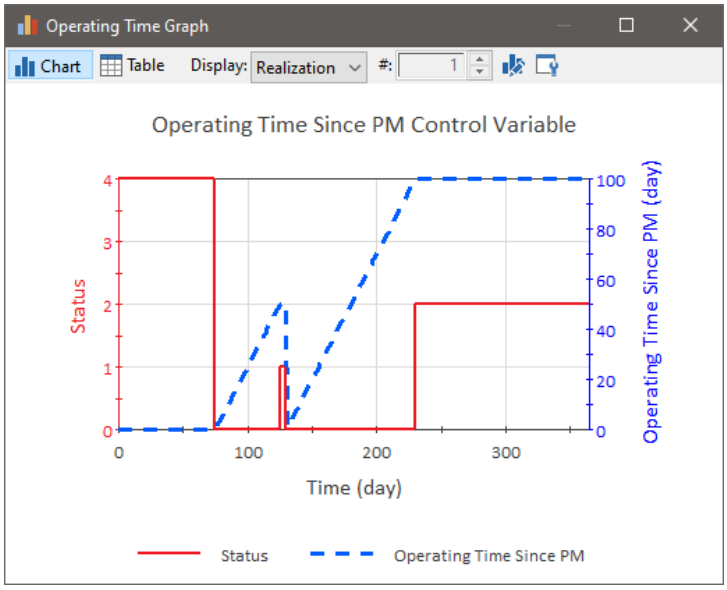
Read more: Failure Modes Available for Function and Action Elements (page 96).

In the plot below, we see the behavior of the element whose failure mode uses a FMCV based on Total time:



You can see that the Total time control variable accrues time from the start of the simulation, so even though the element is only turned on at 75 days, it fails at 100 days.

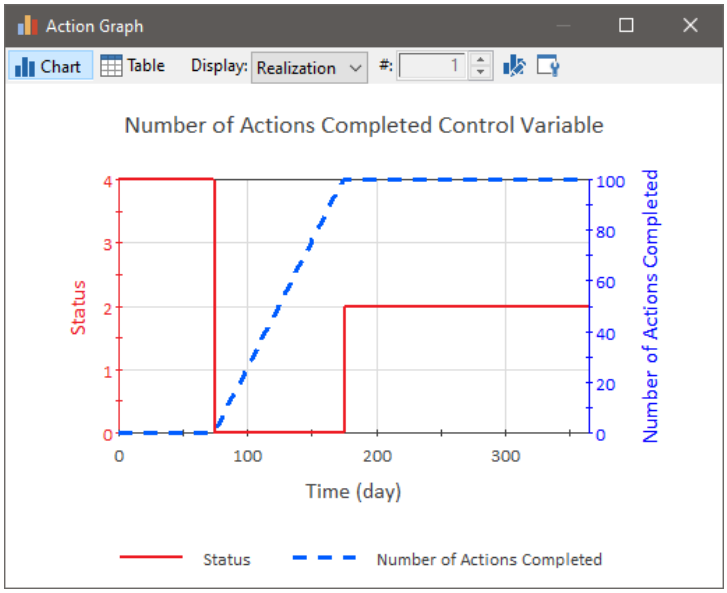
The plot below shows the behavior of the element whose failure mode uses a FMCV based on Operating time:



In this case, you see that the control variable accrues time from 75 to 125 days, when a PM:Preventive Maintenance mode occurs. The control variable value stops accruing time at 125 days by the PM event, and then is reset and starts to accrue time again when the element is placed back in service at 130 days. At 230 days, the element fails, as it has been operating without a PM for 100 days.

Read more: Modeling Maintenance in the Reliability Module (page 114).

The following plot shows the behavior of the Action element whose failure mode uses a FMCV based on Number of completed actions:



From the plot, you can see that this element fails at 175 days, after its action has been triggered 100 times. The Number of Actions Completed control variable is similar to the Total time control variable in that it makes no distinction between the component being operating or off.

Example: Creating User-Defined Base Variables

In some cases, it may be appropriate to define a failure control variable that is defined with respect to a base variable other than time or the number of actions completed (e.g., mileage). Any monotonically increasing function can be specified as a base variable. In most cases, a user-defined base variable will be an Integrator or Reservoir element.

Read more: Creating User-Defined Base Variables (page 110).

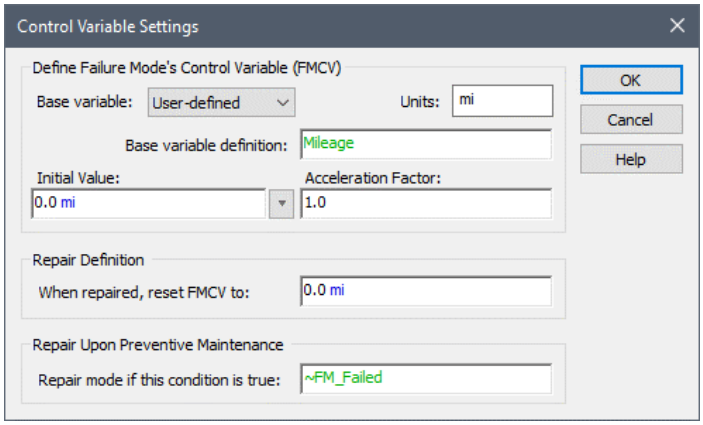
GoldSim samples the change in the control variable required to cause failure, and monitors the change in the variable that you've selected. When the change in the control variable (taking into account Acceleration, and Initial Value settings) exceeds that required to cause failure of the mode, the mode fails.

It is important to note that failure modes using user-defined base variables cannot interrupt the simulation at the exact time of failure like failure modes using time-based control variables. Failures can only occur at a scheduled timestep, or when an event occurs (since GoldSim only evaluates the base variables at these times). This means that models that employ user-defined control variables typically require significantly more timesteps to accurately model a system.

The model file Mileage.gsm, found in the Reliability Examples folder in your GoldSim directory directory (accessed by selecting **File | Open Example...** from the main menu), is a very simple model of a car (one reliability element) that contains a number of failure modes that use the user-defined variable "mileage" as the base variable.

In this case, an Integrator element is used to represent mileage, and a Stochastic element, resampled once per year, is used as the mileage Integrator's rate of change input.

You can see how a User-defined control variable is set up by going to the Failure Modes page of the Car element's property dialog, selecting a failure mode and clicking the **Settings…** button:



Example: Working with Internal and External Requirements

In most simple cases, the location of elements is a matter of logical organization (i.e., they should be placed where the component intuitively should be located). This works for most systems because an equivalent external and internal requirements tree will produce the same result.

Read more: Defining Operating Requirements for Reliability Elements Using Logic Trees (page 64); Modeling a Reliability Element as a System with Child Elements (page 58).

The model file InternalExternal.gsm, found in the Reliability Examples folder in your GoldSim directory directory (accessed by selecting **File | Open Example...** from the main menu), shows a simple system constructed with two identical models: one with internal requirements and a second with external requirements.

In the model, the Internal_Req and External_Req elements both require that either another reliability element (Internal_Reference and External_Reference respectively) with a simple failure rate of 1/3yr is operating (or in the case of the child, operable), or that a Status element is true (the Status_Int and Status_Ext elements become true at 2.5 years).

If you run the model and view plots of availability and reliability in the Summary portion of the **Results** tab for both elements over 1000 realizations, you'll see that the results for both versions of the model are statistically identical for Reliability and Operational Availability:

External Requirements:

Summary

Results for 1000 realizations with mean duration of 5 yr.

Measure	Confidence Bounds		
	5%	Mean	95%
Operational Availability:	0.8300	0.8392	0.8484
Inherent Availability:	1.0000	1.0000	1.0000
Reliability:	0.4092	0.4350	0.4608

Internal Requirements:

Summary

Results for 1000 realizations with mean duration of 5 yr.

Measure	Confidence Bounds		
	5%	Mean	95%
Operational Availability:	0.8300	0.8392	0.8484
Inherent Availability:	0.8300	0.8392	0.8484
Reliability:	0.4092	0.4350	0.4608

The Inherent Availability differs because in the case of the External_Req element, it ceases to operate because of an external requirement. Hence, the element itself is considered *operable*, even though it is not operating.

Read more: Availability and Reliability Results Summary (page 132).

There are some situations where location of a reliability element can be important. The most important of these situations are listed below:

- If Component A can operate even when Component B has failed, A cannot be a child of B (since all child elements stop operating when the parent stops operating).
- Elements that reference the locally available properties of another element as inputs (e.g. ~Status, ~Failed, ~OperatingTime and ~TotalTime), *must* be placed as child elements to the element they are referencing. If you try to reference these locally available properties in a peer element (i.e., without being a child), it will use its own locally available property values instead of the element of interest.

  Read more: Common Reliability Element Locally Available Properties (page 62).
- You cannot create an external requirements reference to the child element of a peer. For example, if Component B is a system (with children B1, B2 and B2), Component A (a peer to Component B) cannot directly refer to B1, B2 or B3 in a requirements tree. Note that this does not prevent you from referencing a peer whose logic tree includes references to child elements (i.e., A could reference B as an external requirement, and B could reference B1 as an internal requirement). It only restricts direct links to a child element of a peer.
- You cannot have two peer elements dependent (directly or indirectly through an intermediate element) on each other in order to operate. Such a system would result in recursive logic.

The model file InternalExternalRestrictions.gsm, found in the Reliability Examples folder in your GoldSim directory directory (accessed by selecting **File | Open Example...** from the main menu), demonstrates some of the restrictions that may influence the decision to place elements as peers or as children. The model will walk you through attempts to violate the restrictions on referencing the child of a peer element, and will also demonstrate what will happen when you try to create a situation where two peer elements are dependent on each other in order to operate.

In addition, the model shows ways that you can work within these restrictions to create your model in cases where your system has these types of situations.

Example Models Illustrating Advanced Reliability Module Concepts

The example models described below demonstrate some of the more advanced reliability module features that you may want to use in your own models.

Example: Modeling Dynamic Failure Mode Behavior Such as Burn-In

In some situations, the behavior of a component (with regard to failure) may change significantly over time. There are two ways to model this type of behavior:

- The parameters of the failure mode can be specified to change over the course of the simulation (i.e., you can vary distribution parameters dynamically); or
- You can choose to model a component's behavior using multiple independent failure modes.

Read more: Defining Failure Modes (page 92).

An example of this type of situation can be found in electronic components. These components typically have a high rate of failure initially (often called a burn-in period), and then a very low rate of failure for a period of time before the failure rate begins to rise again (producing a bathtub shaped failure distribution).

GoldSim does not have a "bathtub shaped distribution" that you can select from the failure mode list. Instead, to model this, we could set up an exponential distribution where the failure rate is a function of the elapsed time of the simulation (initially, the failure rate would be high, then drop to near zero, and increase again at the end of the component's lifespan). Alternatively, we could combine two or more independent failure modes to produce the same sort of behavior.

If we use multiple failure modes, we can make use of a special failure mode, called Defective Component. This failure mode is specifically designed for modeling the burn-in period. It only affects a proportion of components, and if a component is affected, the component fails according to an exponential distribution with a failure rate you specify. Components that are not affected by the Defective Component mode fail according to the remaining failure modes that you've specified, and components that are affected can still fail as a result of any other failure modes you might specify.

The model file BurnInPeriod.gsm, found in the Reliability Examples folder in your GoldSim directory directory (accessed by selecting **File | Open Example...** from the main menu), illustrates this second approach.

In this file, there is a single function element with two failure modes.The first is a Defective Component failure mode, affecting 25% of components, with a failure rate of 2 yr^{-1}. This failure mode represents the failures during the burn-in period. The second failure mode is a normal failure mode with a mean value at failure of 8 years and a standard deviation of 1.3 years, representing end-of-life failures.

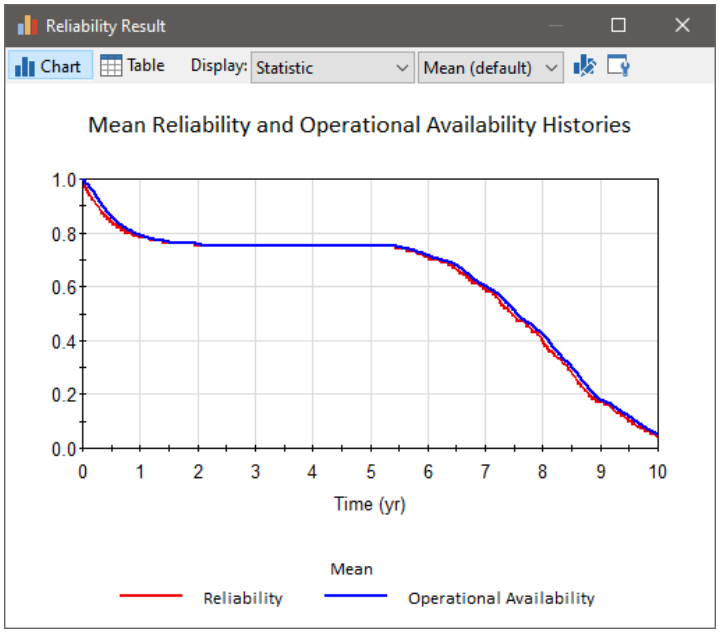
When you run the model and view plots of the Mean Reliability and Availability, you will notice the following:

1. The Mean Reliability and Availability time histories are identical. This is because the failures are not repaired, and hence availability and reliability time histories measure the same thing. (There will be minor differences in the plots because Reliability is computed instantaneously, while Availability is computed based on averaging over a small time period previous to the time point.)

   Read more: Availability and Reliability Histories and Statistics (page 134).

2. You'll see that there is an exponential drop in those metrics almost immediately down to about 75% (the effect of the Defective

Component failure mode). After that, the availability and reliability plateau, until failures begin to occur as a result of the Normal failure mode that has been specified.



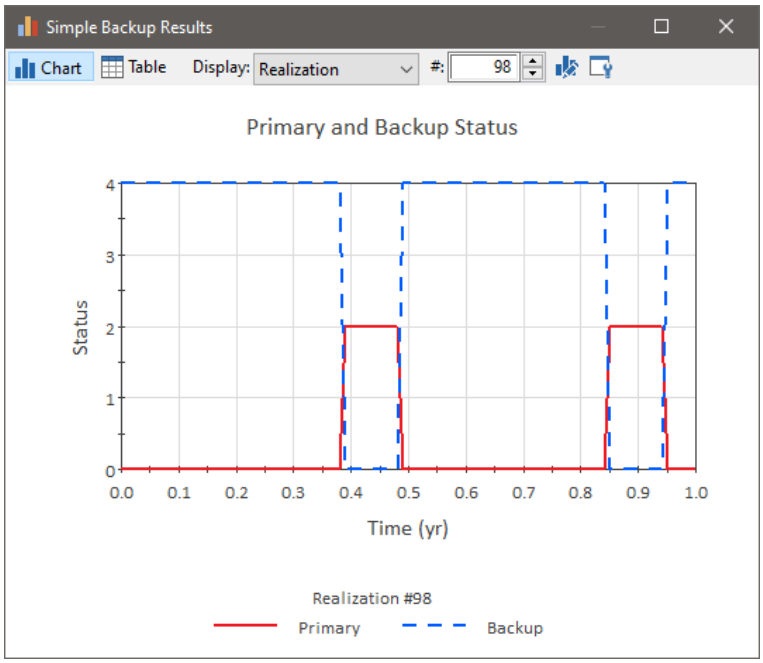
Example: Modeling the Switchover to a Backup Component

Many systems have backup components that can be switched on in the event of a failure of a primary component. There are a number of ways to model the switch to a backup component in GoldSim, but the easiest way is to make use of the StartOperating and StopOperating event outputs of the primary component.

Read more: Common Reliability Element Outputs (page 60).

The model file Backup.gsm, found in the Reliability Examples folder in your GoldSim directory directory (accessed by selecting **File | Open Example...** from the main menu), provides both a simple and a more advanced example of modeling the switchover from a Primary to a Backup component. In both cases, the Primary element has a normally distributed mean time to failure of 6 months, with a mean of 1 month. Failures are repaired according to a Gamma distribution with a mean of 1 month and a standard deviation of 1 week. The Backup element has no failure modes.

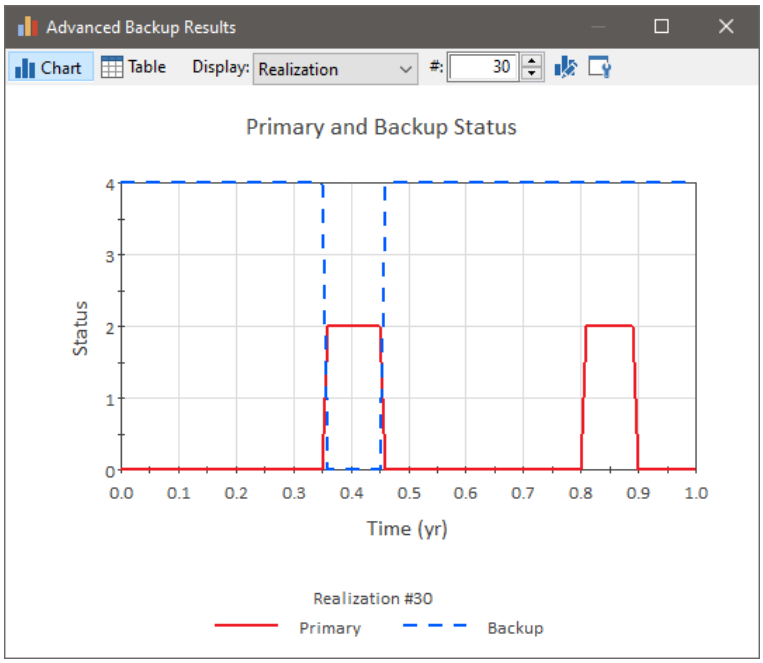
The first (top) example uses the "Stop Operating" output of the primary component to start the standby system. When the primary component is repaired, the primary component's "Start Operating" output is then used to turn the standby system back off. Because the backup component has no failure modes, it means that the backup component is always functional when the primary component fails, as shown in the time history plot of status values below:



The second (bottom) example uses two Action elements to model the control system that turns the backup unit on and off. The control system has an unreliable failure mode with a Reliability of 0.9, meaning that it will successfully turn the backup on or off 9 times out of 10. The Action elements modeling the control system are still triggered by the StopOperating and StartOperating output of the element modeling the Primary system, but the Backup system's On and Off triggers are now linked to the ActionOK outputs of the two Action elements representing the control system.

Read more: Outputs Available Only for Action Elements (page 77).

With this approach, the backup can fail to start, or fail to stop when the Primary system returns to service. The plot below shows a realization where the Backup responds to the first failure correctly, but fails to start the second time the Primary component fails.

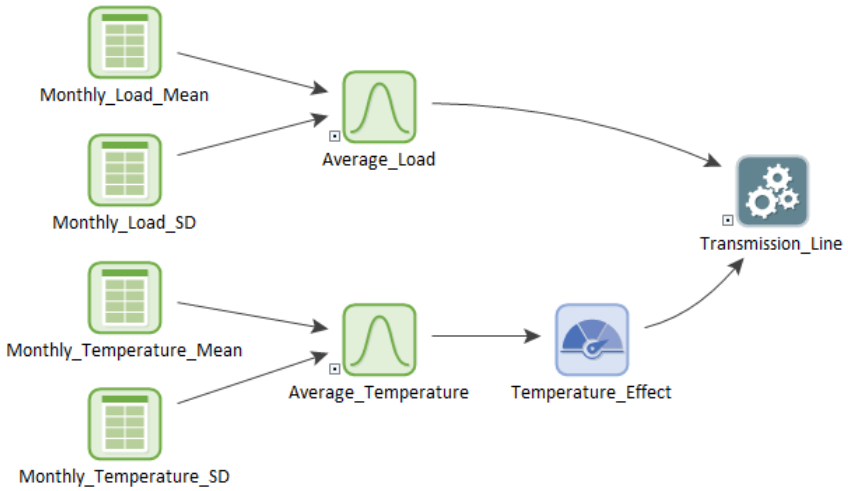


Example: Modeling Changing Operational Environments Using Failure Mode Acceleration

In many cases, a failure mode may be affected by multiple factors. For example, the wear on a turbofan engine may primarily be related to the number of hours it operates, but wear might be accelerated in hot environments. A failure mode's acceleration features can be used to simulate the effects of different operational environments on the control variable by accelerating or decelerating the amount of wear for a given change in the control variable.

Read more: Modeling Acceleration for Failure Mode Control Variables (page 112).

The model file Acceleration.gsm, found in the Reliability Examples folder in your GoldSim directory directory (accessed by selecting **File | Open Example...** from the main menu), is a simple model of a fictitious power transmission line. Its failure is time based, but it depends on the average monthly load on the system and the average monthly temperature. The line typically fails after 35 years (normally distributed with a standard deviation of 5 years) with an average load of 1MW, and mean operating temperatures between 0C and 20C.

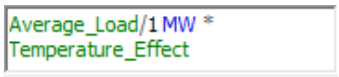


Average loads that are lower (e.g., 500kW) induce lower wear (in this case, the line wears at 50% of the rate it would transmitting 1MW), while average loads that are higher cause higher wear (e.g., transmitting 2MW causes the line to wear twice as fast).

Temperature extremes also cause wear. Cold temperatures and the formation of ice increase wear below 0C (wear is multiplied by a factor of three), and high temperatures cause the lines to sag. As a result, every increase of 10C in the average temperature beyond 20C increases wear on the lines by a factor of 2.

Both the average load and average temperature are resampled each month, using month-by-month data for both values.

In the model, we have a reliability element representing the transmission line. It has a single, normal failure mode that is not repaired. If you view the **Failure Modes** tab, and click the **Settings…** button, you will see that an Acceleration Factor has been specified for this failure mode:



Temperature_Effect is a Selector element (a nested if statement) that is:

- 3 if the average temperature is below 0C;
- equal to the (Average_Monthly_Temperature – 20C)/10K * 2 if the average temperature is greater than 20C and
- 1 otherwise

In this way, we are able to combine the effects of time, load and temperature to get a more accurate picture of how the line would behave, even if it was not transmitting 1MW of energy at a temperature of between 0C and 20C.

Example: Modeling Component Maintenance and Replacement

The behavior of many systems is highly dependent on the maintenance, repair and replacement of components within the system. In addition to the automatic repair option provided for each failure mode, GoldSim provides a number of other powerful ways to model repair, replacement and maintenance.

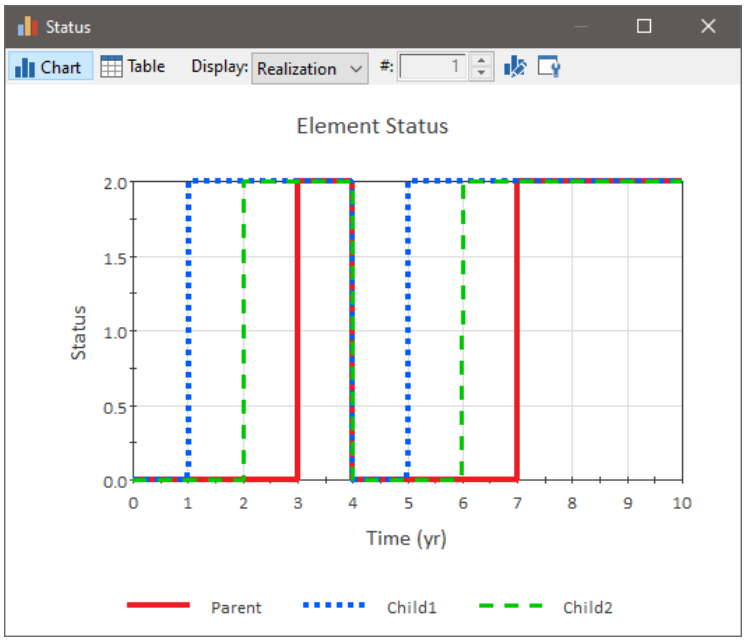
Read more: Modeling Maintenance in the Reliability Module (page 114).

The model file ReplaceTrigger.gsm, found in the Reliability Examples folder in your GoldSim directory directory (accessed by selecting **File | Open Example...** from the main menu), demonstrates the Replace trigger, which instantaneously replaces a component and all of its children with new components.

Read more: Simulating Replacement Using the Replace Trigger (page 120).

In the example, Child_1 has a "Specified Value Exceeded" failure mode that fails at 1 yr. Child_2 is identical, except that it fails at 2 years. The Parent also has a "Specified Value Exceeded" failure mode that fails at 3 years. (Note that the parent does not require the children to be operable in order to operate.)

The statuses of each element over the course of the simulation looks like this:



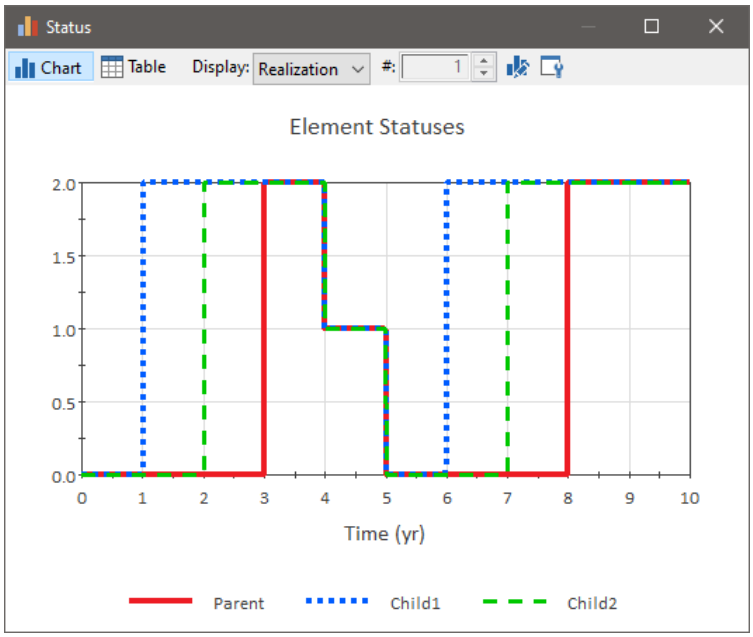
Child_1 fails after 1 year, Child_2 after 2 years, and the Parent after 3 years. The Parent's Replace trigger is activated at 4 years, and immediately replaces the Parent, Child_1 and Child_2. This means that failure occurs again for Child_1 at 5 years, Child_2 at 6 years and the Parent at 7 years.

The model file PreventiveMaintenanceReplacement.gsm, found in the Reliability Examples folder in your GoldSim directory directory (accessed by selecting **File | Open Example...** from the main menu), demonstrates the "PM:Replacement" failure mode, which is nearly identical to the Replace trigger with one key difference: a repair with a stochastic delay time can be specified. In this case, the "repair delay" represents the time it take to replace the component (when the Replace trigger is used, replacement is instantaneous). Note that by definition, any children of an element undergoing a PM:Replacement mode are replaced with a new component.

Read more: Simulating Replacement as a Failure Mode (page 118).

In this model, there are again two children that fail at 1 and 2 years, and the Parent element fails at three years. Replacement is triggered when 4 years of simulated time have passed, and replacement takes exactly one year. (Note that the parent does not require the children to be operable in order to operate.)

The plot of element statuses over the course of the simulation looks like this:



Child_1 fails after 1 year, Child_2 after 2 years, and the Parent after 3 years. The PM:Replacement mode is activated at 4 years, and replaces all three elements with new components at 5 years. This means that failure occurs again for Child_1 at 6 years, Child_2 at 7 years and the Parent at 8 years.

The final method of maintenance modeling illustrated here is the "PM: Preventive Maintenance" failure mode. This is the most complex of the maintenance methodologies. The PM: Preventive Maintenance mode's primary function is to repair other failure modes. The impact of a PM on other failure modes is controlled by the "Repair Following Preventive Maintenance" and "Repair Definition" fields in the Control Variable Settings dialog.

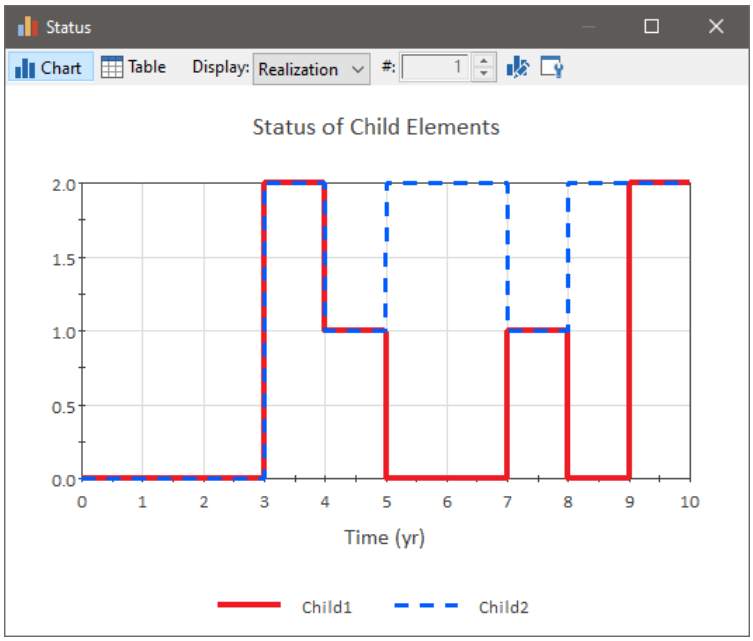
Read more: Simulating Preventive Maintenance as a Failure Mode (page 114).

The model file PreventiveMaintenance.gsm, found in the Reliability Examples folder in your GoldSim directory directory (accessed by selecting **File | Open Example...** from the main menu), demonstrates the "PM:Preventive Maintenance" failure mode In the model, there are again two child elements. Both children have two failure modes. The first is a failure that occurs after 3 years of operating time. The second is a failure mode that occurs after 3.5 years of total time.

Child_1 repairs both failure modes during a PM, while Child_2 only repairs the first failure mode during a PM.

Each child also has a PM: Preventive Maintenance mode that lasts for 1 year and is triggered each time the parent undergoes a PM:Preventive Maintenance mode.

The parent has no failure modes but does have a PM:Preventive Maintenance mode that lasts for exactly 1 year. This mode is triggered at 4 and 7 years.



When the simulation is run, both Child_1 and Child_2 fail due to the first failure mode (based on Operating time) at 3 years. The second failure mode (based on Total time) fails at 3.5 years (i.e., both modes have occurred by this time). The PM:Preventive Maintenance mode of the parent is triggered at 4 years, and this immediately triggers the preventive maintenance of the two children.

Because both failure modes for Child_1 are repaired by a PM , Child_1 starts operating at 5 years (at the completion of the PM). However, Child_2 remains failed (and therefore shows a status of 2), since the second failure mode is not repaired by the PM.

A second PM is performed at 7 years (and both children enter 1 year of PM). For Child_1, this resets the control variable value for the first failure mode (occurring after 3 years of operating time since a PM event). Child_2 remains failed due to failure mode 2 at the conclusion of the PM. Child_1 begins operating again at 8 years, with the 2nd failure mode causing failing at 8.5 years.

Example: Modeling Non-Fatal Failure Modes

By default, any failure modes that are added to a Reliability element are automatically added as Internal Requirements in the element's "Operating Requirements" tree. In particular, GoldSim automatically inserts a "Not" condition in the internal requirements portion of the requirements tree: Not ~Failed[n], where n is the failure mode.

Read more: Failure Modes and Internal Requirements (page 95).

This implies that the failure mode is assumed to be fatal to the component (i.e., if the failure mode occurs, the component itself fails and is no longer operative). That is, by default, whenever you add a failure mode, GoldSim treats it as if it is a fatal failure mode.

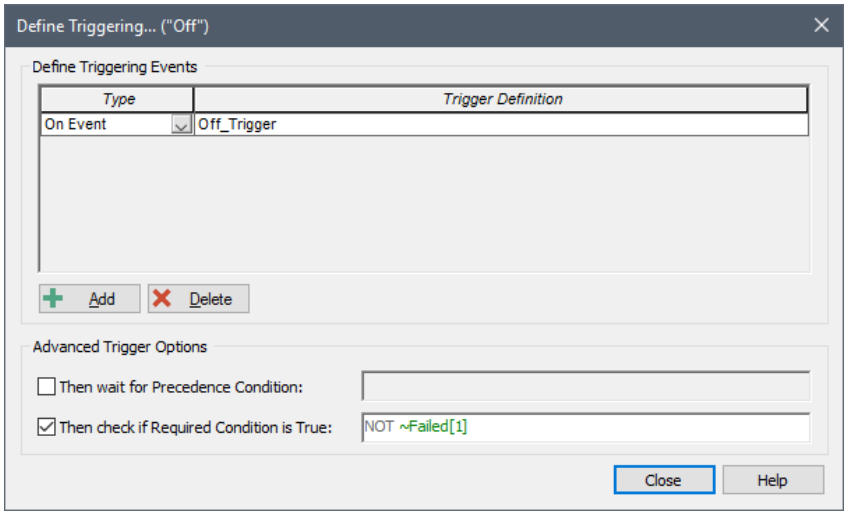
In some cases, however, the failure mode may not be fatal to the component (i.e., when the failure mode occurs, it does not cause the component itself to fail). This type of situation can be modeled by simply removing references to non-fatal failure modes from the Requirements tree.

Read more: Adding, Removing and Editing Nodes in the Logic-Tree (page 68).

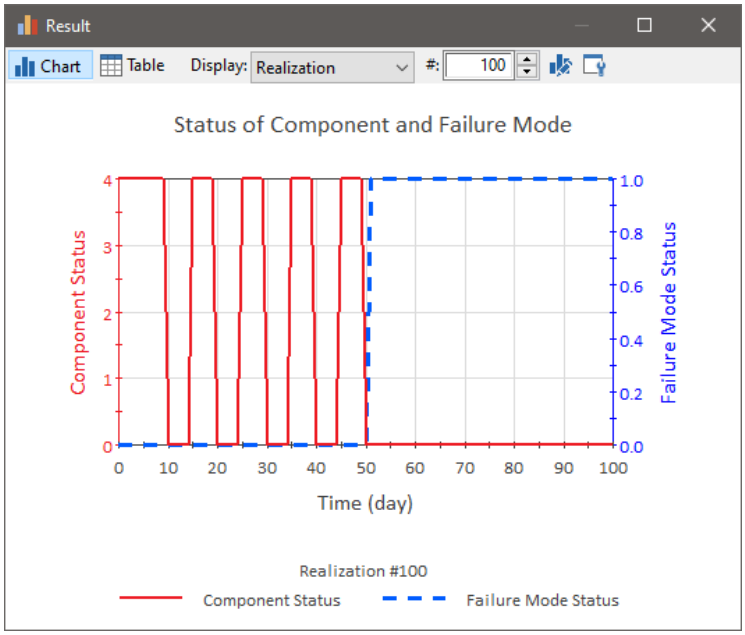
Let's consider two examples.

First, let's consider a case where you are modeling a component that gets turned On and Off, and a particular failure mode causes the component to fail On (i.e., it continues to operate, but it is "stuck" On and cannot be turned Off).

The model file FailsOn.gsm, found in the Reliability Examples folder in your GoldSim directory directory (accessed by selecting **File | Open Example...** from the main menu), illustrates such a case. In this model, the component is turned on every ten days, and turned off 5 days after being turned on. A single failure mode is defined. It is not fatal to the component (it has been removed from the Internal Requirements). However, when defining the Off trigger for the component, the failure mode is directly referenced as a Required Condition:



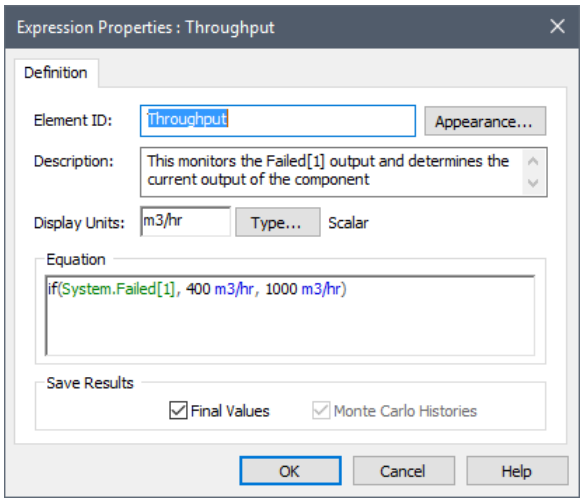
In the realization below, you will see that the component successfully turns on and off several times, but due to the occurrence of a failure around 50 days, it is "stuck" in the On position and never turns off again:



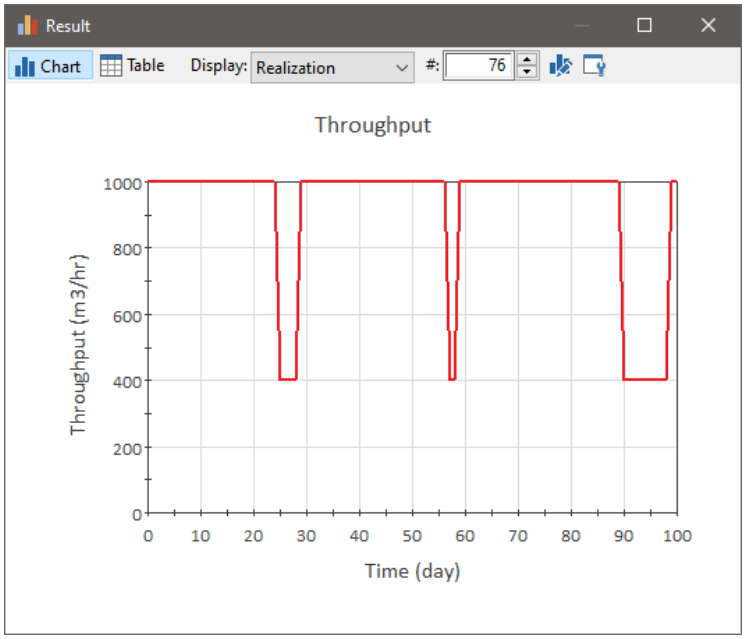
Now let's consider second example where a failure, while not fatal, reduces the capacity or performance of some internal component of the system. The model file FailedOutput.gsm, found in the Reliability Examples folder in your GoldSim

directory directory (accessed by selecting **File | Open Example...** from the main menu), illustrates such a case.

In this model, a component has a single failure mode that is not fatal, but it does impact the throughput of material through the component. This is done by referencing the failure state variable in an If statement describing the throughput:



The failure mode reduces the throughput from 1000 m3/hr to 400 m3/hr. The failure is assumed to be exponential with a rate of once every 30 days. It is repaired in approximately 5 days. A typical realization of the throughput is shown below:

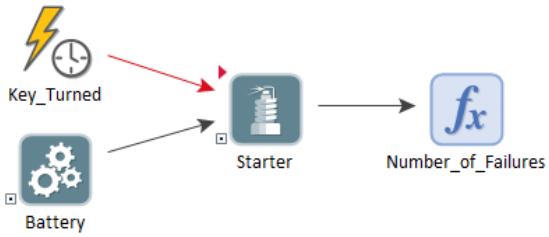


Example: Handling Actions Internally

An option for Action elements is to handle their actions internally. This means that when a parent Action element's action is triggered, it causes a series of events in that element's children to occur, in effect triggering a chain of events inside the component. The **OK** and **Failed** triggers in the parent element monitor the behavior of the child elements and issue an ActionOK or ActionFailed event.

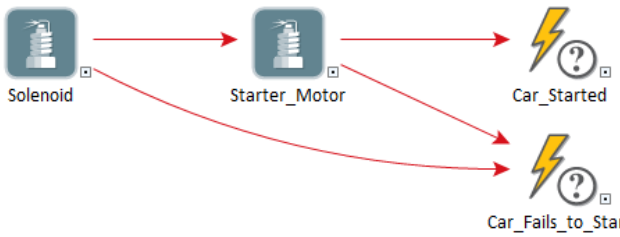
Read more: Handling Actions Internally (page 125).

Typically, this type of approach will be used to model mechanical systems where a number of steps have to be successfully completed in sequence before a particular action can be declared successful:



The model file InternallyHandledAction.gsm, found in the Reliability Examples folder in your GoldSim directory directory (accessed by selecting **File | Open Example...** from the main menu), is of a car starter, with a Timed Event simulating the turning of the ignition key. The car is started three times per day, and the starter requires an operating battery, as well as an operable solenoid and motor to operate.

When the key is turned, this triggers the Starter's action. Because the Starter is modeled as a system, and the **Handle action internally** option is selected, child Action elements of the Starter have access to a unique type of Automatic triggering called "Auto Action." This triggers the child component's action whenever the parent component's action is triggered:



The Starter has two Action component children, Solenoid and Starter_Motor. Each of these elements has two failure modes: an Unreliable failure mode, and a standard failure mode representing wearout. It is important to note that the Unreliable failure mode is unique in that it does not cause failure of the element, but rather causes triggered actions to fail a specific portion of the time.

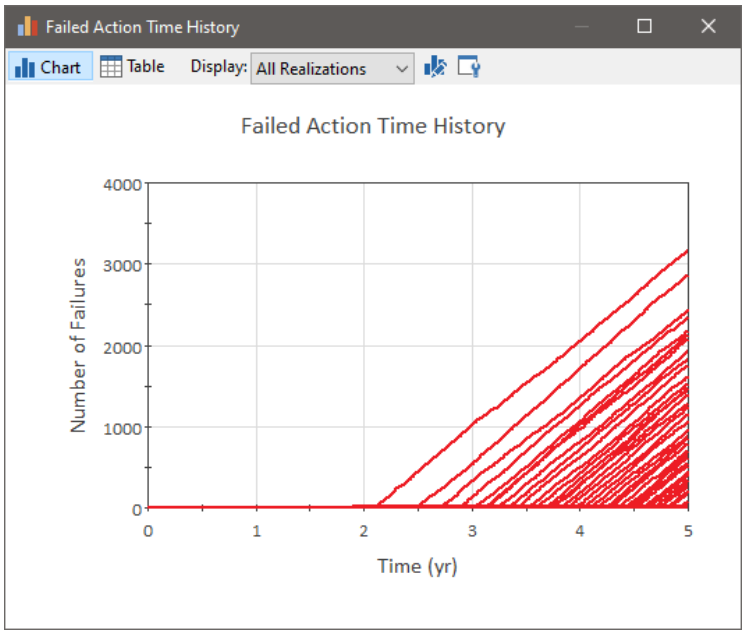
Read more: Failure Modes Available Only for Action Elements (page 99).

The Starter's action triggers the action of the Solenoid element. If the Solenoid completes its action successfully, its ActionOK event output triggers the Starter_Motor's Action trigger. If unsuccessful, it triggers the Car_Fails_to_Start Triggered Event, and the Starter_Motor's action is not triggered.

If the Starter_Motor is triggered, and its action is successful, the Starter_Motor's ActionOK event output triggers the Car_Started element. If it is unsuccessful, the Car_Fails_to_Start element is triggered. The Car_Started element and the Car_Fails_to_Start element are linked to the **OK** and **Failed** triggers of the parent Starter element.

Note: It is very important that each Auto Action trigger should only lead to a single **OK** or **Failed** conclusion. GoldSim is unable to determine which Action trigger corresponds with which OK or Failed result, so systems where multiple OK and/or Failed results are generated will in most cases lead to undesired behavior.

When the model is run, the time history of the number of unsuccessful actions shows that the starter first begins to fail after about two years in service:



Example: Using Custom Reliability Outputs to Report Throughput Calculations

In some cases, to make your model logic more transparent, you may want to define your own "custom" outputs for a Reliability element. These outputs can be functions of local properties of the Reliability element itself (e.g., it's Status), or, if the element is being modeled as a system, they may be outputs (or functions of outputs) of elements *inside* the Reliability element (e.g., a throughput if the element represented some processing step).

To facilitate this, GoldSim allows you to create these custom outputs via an Output Interface Definition section on the **Results** tab of Reliability elements.

Read more: Defining and Using the Output Interface for a Reliability Element (page 142).

The model file Throughput.gsm, found in the Reliability Examples folder in your GoldSim directory directory (accessed by selecting **File | Open Example...** from the main menu), provides an example of such an application.

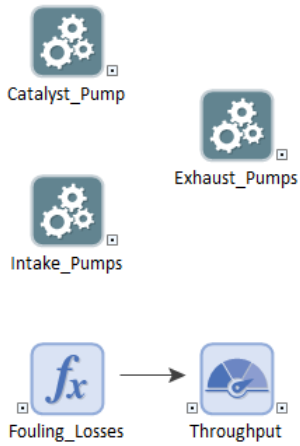
The model is of a process tank where materials from a prior process are mixed with a catalyst. The product of the chemical reaction between the components is then pumped out of the tank. The chemicals involved tend to "foul" the tank and other components with deposits as the process is running. This reduces throughput from nominal by a rate of 10% for every year since a preventative maintenance event was undertaken.

The tank is also equipped with an agitator that increases throughput (by mixing the three components more thoroughly). The process will still produce product

if the agitator fails, but at a much lower rate. The agitator is replaced during a yearly preventative maintenance event if it has failed.

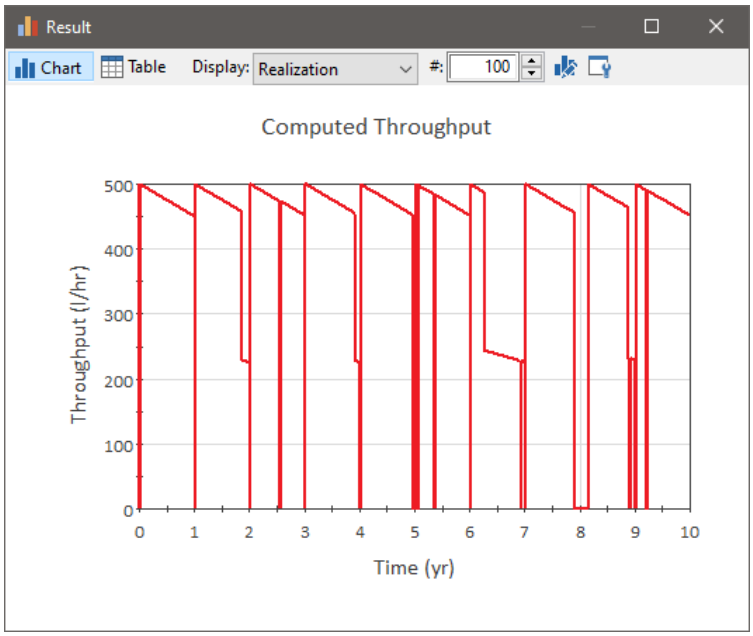
Child elements within the parent element (i.e., the Tank) are used to represent the behavior of the pumps. The intake, exhaust and catalyst pumps must all be functional for the process to be successful. The Tank itself has two failure modes (a fatal, structural failure mode, and a non-fatal failure mode for the agitator) plus a preventative maintenance event triggered at the start of each year.

Within the parent element, in addition to the three pumps, there are two elements that track the degree of fouling, as well as the throughput:



The Throughput is computed as a function of the fouling losses and the current operating state of the system. In particular, if the Tank has fatally failed (due to a pump failure, or a structural failure), the throughput is zero. If only the agitator has failed, the throughput is reduced by 50%. If there are no failures, the throughput is the design throughput for the process.

The output is referenced directly as an output of the Tank element (as "Process_Tank.Throughput"), and a single realization is plotted below:



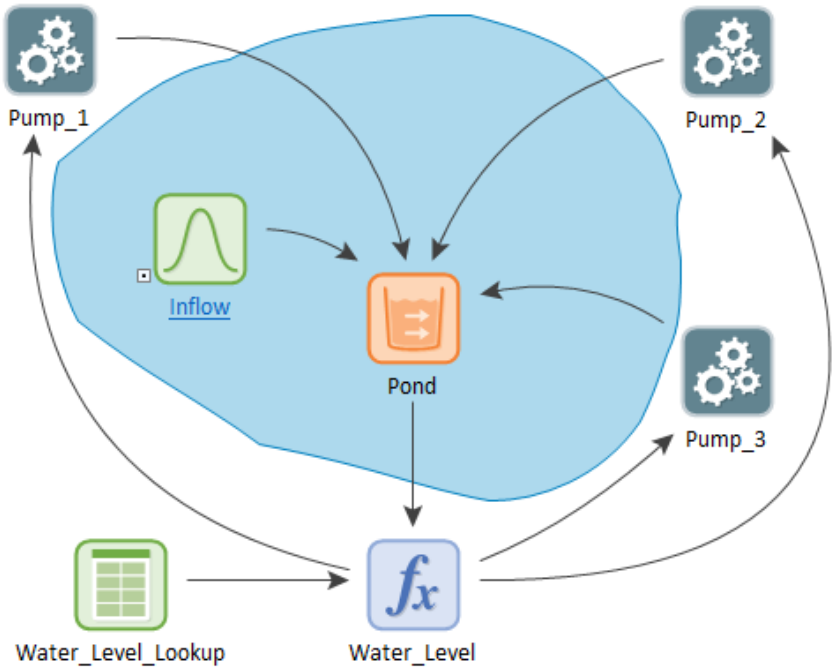
As can be seen, the throughput starts at 500 l/hr, slowly decreases due to fouling, occasionally decreases by 50% due to agitator failure, and sometimes goes to zero, due to either fatal failures or preventive maintenance. A preventive maintenance returns the throughput to 500 l/hr.

Example Models Illustrating the Use of Basic GoldSim Elements with the Reliability Module

The example models described below demonstrate how the Reliability Module can complement and make use of some of the basic GoldSim elements.

Example: Using Reliability Elements to Model Failing Pumps

The model file Pond.gsm, found in the Reliability Examples folder in your GoldSim directory directory (accessed by selecting **File | Open Example...** from the main menu), is a simple model of a pond where the amount of water is controlled by three pumps. Water flows into the pond at a rate of 400m^3/day, and it is pumped out of the pond by the three pumps.



In this model, we have a pond with an inflow rate of approximately 600 m3/day (the inflow is sampled daily from a Normal distribution with a mean of 600 m3/day and a SD of 100 m3/day).

Three pumps are used to remove this water. Each pump can only be operated if there is a sufficient depth of water in the pond (which varies from pump to pump). Pump_1 can only operate if there is more than 1.0m of water in the pond. Pump_2 can only operate if there is more than 2.0m of water in the pond. Pump_3 can only operate if there is more than 2.5m of water in the pond. The check to ensure sufficient water depth is done using a Condition type node in the External Requirements tree referencing the Water_Level element.

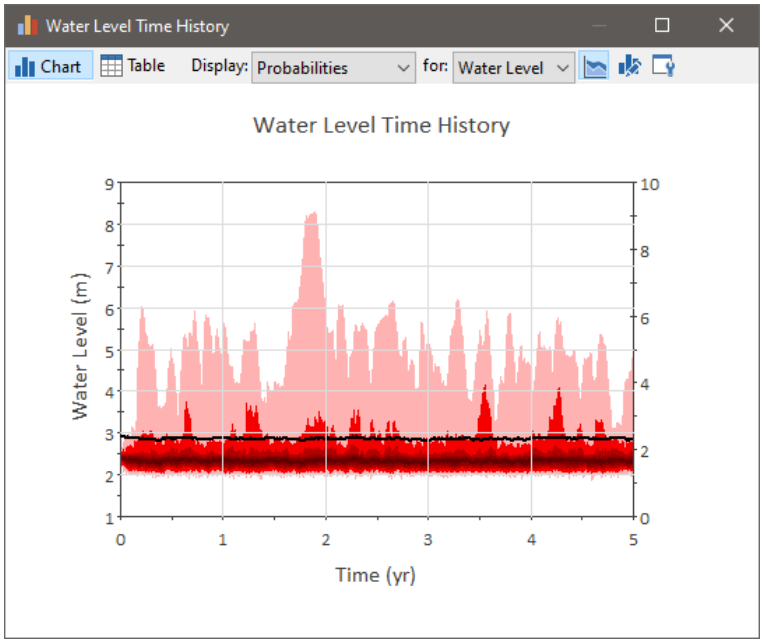
All three pumps can remove 300 m3/day and have an Exponential/Poisson failure mode with a failure rate of 1/3yr that is automatically repaired according to a Gamma distributed delay with a mean of 30 days and standard deviation of 10 days. The calculation of the water removed from the pond by the pumps is

done using a series of if/then/else statements referencing the condition of each pump. For example, the contribution of Pump_1 is calculated using the following if/then/else statement:

if (Pump_1 = RL_Operating, 300 m3/day, 0 m3/day)

The result of this statement is added to the result of similar statements for Pump_2 and Pump_3 to determine the Rate of Withdrawal from the Pond element.

The time history of the water level in the pond over the course of the simulation looks like this:



The plot shows that for the most part, the level of the pond is controlled at about 2.4 meters. However, spikes as high as 8 meters can occur when the pumps fail.

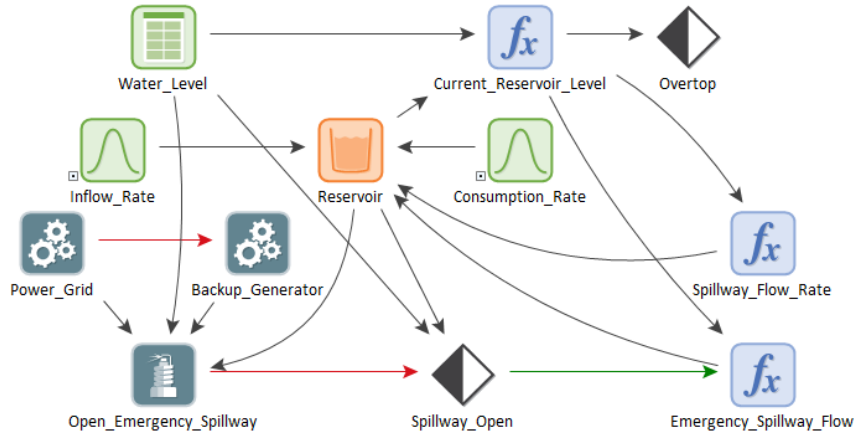
Example: Using Reliability Elements for a Dam Risk Assessment

It is not necessary to use the reliability elements solely to determine the reliability or availability of a particular system. In many cases, a risk analysis model will need to model systems or components that could fail and either directly or indirectly lead to the consequence you are interested in.

Typically, these types of models will use the status output of a Function component (e.g., if a pump is operating then a certain amount of water is removed), the ActionOK/ActionFailed outputs of an Action element (e.g., to determine whether a control action was successful), or the StopOperating output to determine when a critical component goes offline.

Read more: Common Reliability Element Outputs (page 60); Outputs Available Only for Action Elements (page 77).

The model file Overtop.gsm, found in the Reliability Examples folder in your GoldSim directory directory (accessed by selecting **File | Open Example...** from the main menu), is a simple risk analysis/assessment.

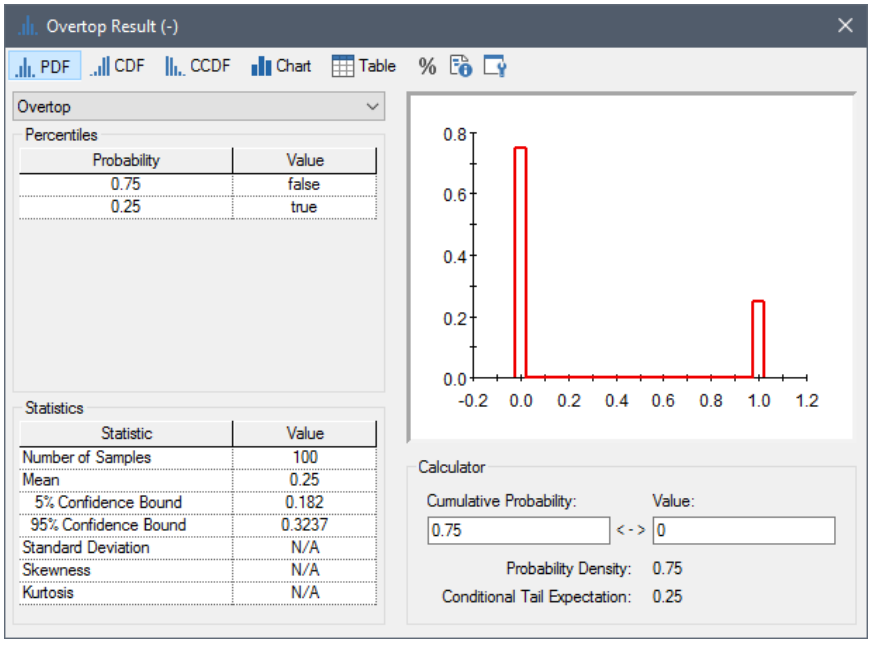


In this model, a water reservoir has a stochastic inflow and consumption rate. It also has two spillways: a passive spillway that begins to drain the reservoir when the water level reaches 16 feet, and an emergency spillway that is opened when the water level reaches 17.5 feet, and is closed when it returns to 16 feet. The flow rates out of the spillways are dependent on the amount of water in the reservoir.

The opening of the emergency spillway is modeled with an Action element called Open_Emergency_Spillway that is triggered when the water level reaches 17.5 feet. It requires that either the Power_Grid function element or the Backup_Generator function element is operating. The Open_Emergency_Spillway, Power_Grid and Backup_Generator elements all have exponential failure modes that are automatically repaired.

The reservoir overtops with potentially catastrophic consequences when the water level reaches 20ft.

The model is run for 5 years, and the probability of the reservoir spilling its banks at least once during that period can be viewed by double clicking on the "Overtop Result" Result element:



It shows that there is a 25% probability that the dam's reservoir would overtop during a given five year period.

The model could be used to evaluate the impact on the probability of overflow of larger spillways, changing the level at which the passive spillway begins to bypass the dam, and changing the levels at which the emergency spillway activates and deactivates.

Example: Modeling Resource Requirements for Reliability Elements

Sometimes desired actions can be constrained by Resource availability. For example, if a system component failed, its repair might not be possible until a mechanic and an appropriate spare part were available. In this case, the pool of mechanics and the stock of spare parts would be considered as Resources. Some resources, such as spare parts, are consumed when they are used. Others, such as mechanics, are only borrowed, and become available again once they are no longer required.

One of the advanced features in GoldSim is the ability to create and interact with Resources. Within GoldSim, a ***Resource*** is defined as something that has a limited supply (e.g., spare parts, fuel, skilled personnel, money) and is required in order for elements of the modelled system to carry out certain actions.

Read more: Modeling Resources in the Reliability Module (page 121).

The model file ReliabilityResources.gsm, found in the Reliability Examples folder in your GoldSim directory directory (accessed by selecting **File | Open Example...** from the main menu), demonstrates the use of Resources when within Reliability elements. In particular, this model creates two (global) Resources Stores: Spares and Technician. The Stores start with 5 Spares and 1 Technician.

Three components are simulated, and each one has a failure mode that is automatically repaired. In each case, however, the repair uses 1 Spare and borrows 1 Technician.

Moreover, the model includes logic for ordering new Spares.

Appendix A: Mathematical Details of the Reliability Module

> Mathematics is a dangerous profession; an appreciable proportion of us go mad...
>
> J.E. Littlewood, *A Mathematician's Miscellany*

Appendix Overview

This appendix describes the mathematical details of the features incorporated into the Reliability Module.

In this Appendix

This chapter discusses:

- Determining When Failures Occur
- Details of Failure Modes Available to the Function and Action Elements
- Details of Failure Modes Available Only to the Action Element
- Details of Repair Time Distributions
- Computing Failure Distributions
- Determining the Root Causes of Unmet Internal or External Requirements
- Calculation of Action Element Delays

Determining When Failures Occur

For each failure mode that has a control variable, GoldSim samples and stores its failure value at the start of the simulation, and resamples it whenever the failure mode is repaired or the element is replaced. The current value of the control variable is re-initialized for replacement, and can be reset for repairs.

As the simulation progresses, the value of the mode's control variable is updated at each time step, accounting for the impact of any specified acceleration factor. The mode fails when the control variable equals or exceeds the sampled value at failure.

For time-based failure modes, GoldSim schedules an event in the model's time queue for the anticipated time of failure. This ensures that regardless of the planned timesteps, the failure will occur at the correct time. If the anticipated time of failure changes due to an acceleration factor or, for an Exponential/Poisson mode, a dynamically changing failure rate, the event's anticipated time is updated.

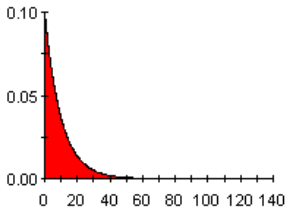
However, it is important to note that the values of acceleration factors and Exponential/Poisson failure rates are only updated when a timestep or an event occurs. If you are modeling an acceleration rate or Exponential/Poisson failure rate that can vary dynamically, you will need to specify a sufficient number of timesteps to accurately model changes in those parameters.

Similarly, the values of user defined control variables are only updated at timesteps or when an event occurs. This means that failure will occur the next time a model is updated (due to a timestep or an event-triggered update) after its control variable exceeds the sampled failure value.

Details of Failure Modes Available to the Function and Action Elements

Simple Failure Rate

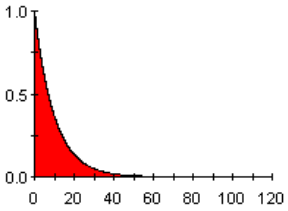
The simple failure rate is the default failure mode for both the Function and the Action element. It is equivalent to the Exponential/Poisson failure mode, and uses Total time as its control variable. This mode cannot be repaired automatically; it can only be repaired using the Replace trigger. The probability distribution function of the underlying Exponential/Poisson distribution has the following shape and equation:



$$f(x) = \frac{1}{\mu} e^{\frac{-x}{\mu}}$$

where μ is the mean control variable value at failure

The reliability function of the underlying Exponential/Poisson distribution has the following shape and equation:



$$R(x) = e^{\frac{-x}{\mu}}$$

The simple failure rate uses Total time as its control variable, and places the time to the next failure in the event queue, which will interrupt the simulation at the exact time of failure.

Because the Exponential distribution is memoryless, the simple failure rate parameter is fully dynamic, and control variable values at failure are updated whenever the Failure Rate changes. The simple failure rate does not support acceleration (except through changes in the failure rate).

Parameters:

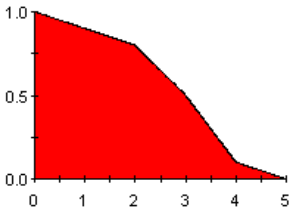
PARAMETER	FULLY DYNAMIC	UPDATED WHEN REPAIRED/REPLACED	STATIC
Failure Rate (μ)	✓		

Advanced Settings:

SETTING	SUPPORTED?
Operating time and Total time control variables	**Total time only**
User-defined control variable	**No**
Number of Actions control variable	**No**
User-defined / Uncertain initial control variable value	**No**
Acceleration	**No**
Repair to Age	**No**

Cumulative Failure Mode

The Cumulative failure mode allows you to specify a table of the value of the control variable and the corresponding probability of surviving to that value. The Cumulative failure mode is actually specified as a reliability function, rather than as a probability density function (you specify the control variable value and the probability of survival). An example of a cumulative failure mode reliability function and the cumulative failure mode reliability equation is shown below.



$$R(x) = 1 \qquad x \le x_1$$

$$R(x) = p_i - (p_i - p_{i-1})\frac{x - x_i}{x_{i+1} - x_i} \qquad x_1 < x < x_n$$

$$R(x) = p_n \qquad x \geq x_n$$

where x_i is the ith control variable value specified and p_i is the corresponding probability of survival.

The Cumulative failure mode supports all of the failure mode control variables. The value of the control variable at the time of failure is randomly sampled from the distribution that you have created. Note that you must supply a control variable value where the probability of survival is 1, but you do not need to provide a control variable value where the probability is zero. If no zero entry is specified, and a probability value is sampled that exceeds the last specified probability value, the failure mode will not occur. The distribution will only be resampled after the failure mode is repaired, or the component has been replaced.

While the Cumulative distribution supports initial control variable values, acceleration and repair to age features, the sets of control variable values and probabilities of survival cannot be changed over the course of the simulation.

Parameters:

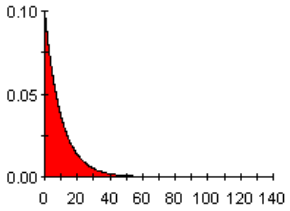
PARAMETER	FULLY DYNAMIC	UPDATED WHEN REPAIRED/REPLACED	STATIC
Control Variable Values and Survival Probabilities			✓

Advanced Settings:

SETTING	SUPPORTED?
Operating time and Total time control variables	**Yes**
User-defined control variable	**Yes**
Number of Actions control variable	**Yes**
User-defined / Uncertain initial control variable value	**Yes**
Acceleration	**Yes**
Repair to Age	**Yes**

Defective Component Failure Mode

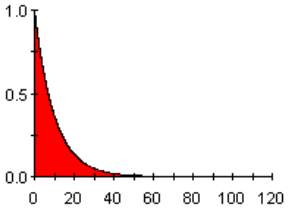
The Defective Component failure mode is a two part failure that requires two arguments: the "Probability of Defect" and the "Rate of Failure if Defective." When GoldSim evaluates a Defective Component failure mode, it first determines whether or not the component is affected by sampling a uniform distribution. If the component is affected, it fails according to an Exponential/Poisson failure mode (the "Rate of Failure if Defective" is identical to the Exponential/Poisson failure mode's "Failure Rate"). The probability distribution function of the underlying Exponential/Poisson distribution has the following shape and equation:



$$f(x) = \frac{1}{\mu} e^{\frac{-x}{\mu}}$$

where μ is the mean control variable value at failure

The reliability function of the underlying Exponential/Poisson distribution has the following shape and equation:



$$R(x) = e^{\frac{-x}{\mu}}$$

Because the Defective Component mode uses the Exponential/Poisson failure mode as a basis, user-defined/uncertain initial values, and repair to age options have no effect, and thus are not supported. Acceleration is supported for components affected by the Defective Component failure mode, and since it uses the Exponential/Poisson failure mode as a basis, the baseline failure rate is multiplied by the acceleration factor. For example, if an acceleration factor of 3 is specified, and a failure normally occurs 5 times per year, GoldSim will use a failure rate of 15 1/yr.

If the component is not affected by the defect, the Defective Component failure mode cannot occur until the failure mode is repaired or the component is replaced.

Parameters:

PARAMETER	FULLY DYNAMIC	UPDATED WHEN REPAIRED/REPLACED	STATIC
Probability of Defect		✓	
Rate of Failure if Defective (μ)	✓		

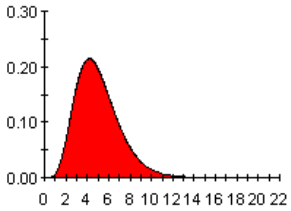
Advanced Settings:

SETTING	SUPPORTED
Operating time and Total time control variables	**Yes**
User-defined control variable	**Yes**
Number of Actions control variable	**Yes**
User-defined / Uncertain initial control variable value	**No**
Acceleration	**Yes**
Repair to Age	**No**

Erlang Multi-Failure Mode

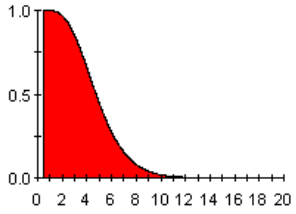
The Erlang multi-failure mode (also known as the Gamma distribution) is another derivative of the Exponential/Poisson failure mode and describes the time to the *k*th failure of *k* identical components which fail according to an Exponential/Poisson distribution. It requires two arguments – the "Number of components" and the "Rate of failure per component."

In the Erlang multi-failure mode, the first component is assumed to operate until it fails, at which time it is replaced by the next identical component. This continues until all of the identical components have failed. Each identical component fails according to an Exponential failure mode. The PDF of the Erlang function has the following equation and shape:



$$f(x) = \frac{\frac{1}{\mu}\left(\frac{x}{\mu}\right)^{k-1}}{(k-1)\,!}\, e^{\frac{-x}{\mu}}$$

where μ is the mean time to failure of a single component, and k is the number of identical components.



The reliability function of the Erlang distribution has the following shape and equation:

$$R(x) = 1 - \frac{\Gamma\left(k, \frac{x}{\mu}\right)}{\Gamma(k)}$$

where $\Gamma(i)$ is the complete gamma function and $\Gamma(i, j)$ is the incomplete gamma function.

Since the Erlang failure mode is an aggregate of a number of Exponential distributions, it is not memoryless. This means that user-defined/uncertain initial values, repair to age, and acceleration options can be used, but it also means that distribution parameters are only refreshed when the component is repaired or replaced.

Parameters:

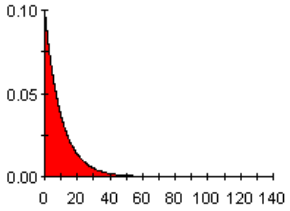
PARAMETER	FULLY DYNAMIC	UPDATED WHEN REPAIRED/REPLACED	STATIC
Number of Components (k)		✓	
Rate of Failure per Component (μ)		✓	

Advanced Settings:

SETTING	SUPPORTED?
Operating time and Total time control variables	**Yes**
User-defined control variable	**Yes**
Number of Actions control variable	**Yes**
User-defined / Uncertain initial control variable value	**Yes**
Acceleration	**Yes**
Repair to Age	**Yes**

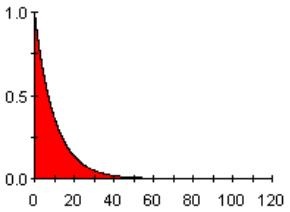
Exponential/Poisson Failure Mode

The Exponential/Poisson failure mode accepts a Failure Rate (Number of Failures/Change in Control Variable) as its only parameter. The PDF of the Exponential function has the following shape and equation:



$$f(x) = \frac{1}{\mu} e^{\frac{-x}{\mu}}$$

The reliability function of Exponential/Poisson distribution has the following shape and equation:



$$R(x) = e^{\frac{-x}{\mu}}$$

The Exponential/Poisson failure mode supports all of the failure mode control variables. The value of the control variable at the time of failure is randomly sampled from the distribution shown above.

Because the Exponential distribution is memoryless, the Failure rate parameter is fully dynamic. However, the memoryless nature of the Exponential distribution means that user-defined/uncertain initial values, and repair to age options have no effect, and thus are not supported.

Acceleration is still supported for the exponential failure mode, but when used with an exponential failure mode, the baseline failure rate is multiplied by the acceleration factor. For example, if an acceleration factor of 3 is specified, and a failure normally occurs 5 times per year, GoldSim will use a failure rate of 15 1/yr.

Parameters:

PARAMETER	FULLY DYNAMIC	UPDATED WHEN REPAIRED/REPLACED	STATIC
Failure Rate (μ)	✓		

Advanced Settings:

SETTING	SUPPORTED	NOT SUPPORTED
Operating time and Total time control variables	✓	
User-defined control variable	✓	
Number of Actions control variable	✓	
User-defined / Uncertain initial control variable value		✓
Acceleration	✓	
Repair to Age		✓

Event-triggered Failure Mode

The Event-triggered failure mode allows you to specify a triggering event or condition that could result in failure, along with a "Probability of failure" should the event or condition occur.

The Event-triggered failure mode does not support control variables – it must be explicitly triggered by a condition or event in GoldSim. When this occurs, GoldSim uses the "Probability of failure" to determine whether or not the event or condition caused the failure of the reliability element. The "Probability of failure" can be dynamic and is evaluated at the time that the Event or condition occurs.

Because the failure mode does not use failure mode control variables, initial control variable values, acceleration and repair to age features, are not supported. the sets of control variable values and probabilities of survival cannot be changed over the course of the simulation.

Parameters:

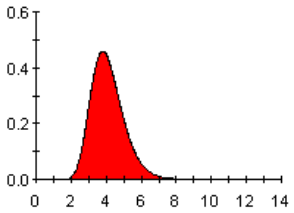
PARAMETER	FULLY DYNAMIC	UPDATED WHEN REPAIRED/REPLACED	STATIC
Probability of failure	✓		

Advanced Settings:

SETTING	SUPPORTED?
Operating time and Total time control variables	**No**
User-defined control variable	**No**
Number of Actions control variable	**No**
User-defined / Uncertain initial control variable value	**No**
Acceleration	**No**
Repair to Age	**No**

Lognormal Failure Mode

The Lognormal failure time distribution has the following shape and equation:



$$f(x) = \frac{1}{\sigma\ x\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\ln x - \mu}{\sigma}\right)^2}$$

where μ and σ are the mean and standard deviation of the logarithm of failure times.

In GoldSim, μ and σ are not specified directly. Instead, you can choose to describe the failure rate distribution using a true mean and standard deviation or a geometric mean and standard deviation.

The true mean is actually equal to:

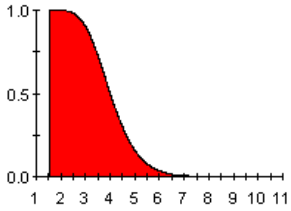
$Mean = e^{\left(\mu + \frac{\sigma^2}{2}\right)}$ and the true standard deviation is equal to:

$$SD = \sqrt{(e^{\sigma^2} - 1)\ e^{2\mu + \sigma^2}}$$

If you choose to specify a true mean and true standard deviation, GoldSim uses the above equations to solve for μ and σ.

The geometric mean and geometric standard deviation are simply equal to e^{μ} and e^{σ}. Again GoldSim solves for μ and σ.

The reliability equation for the lognormal distribution has the following shape and equation:



$$R(x) = \frac{1}{2}\left(1 - erf\left(\frac{\ln(x) - \mu}{\sigma\sqrt{2}}\right)\right)$$

The lognormal distribution supports initial value options, repair to age and acceleration features. Both parameters can be dynamic, but the value of the control variable at failure is only recalculated when the component is placed in service, replaced, or the failure mode repaired.

Parameters:

PARAMETER	FULLY DYNAMIC	UPDATED WHEN REPAIRED/REPLACED	STATIC
Mean (true or geometric)		✓	
Standard deviation (true or geometric)		✓	

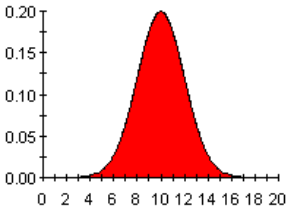
Advanced Settings:

SETTING	SUPPORTED?
Operating time and Total time control variables	**Yes**
User-defined control variable	**Yes**
Number of Actions control variable	**Yes**
User-defined / Uncertain initial control variable value	**Yes**
Acceleration	**Yes**
Repair to Age	**Yes**

Normal Failure Mode

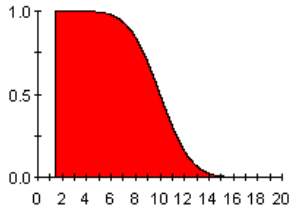
The normal distribution accepts two parameters - the mean and standard deviation of the control variable value at the time of failure.

The PDF of the lognormal distribution has the following shape and equation:



$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

The reliability equation for the normal distribution has the following shape and equation:



$$R(x) = \frac{1}{2}\left(1 - erf\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right)$$

The Normal distribution supports initial value options, repair to age and acceleration features. Both parameters can be dynamic, but the value of the control variable at failure is only recalculated when the component is placed in service, replaced, or the failure mode repaired.

Parameters:

PARAMETER	FULLY DYNAMIC	UPDATED WHEN REPAIRED/REPLACED	STATIC
Mean		✓	
Standard deviation		✓	

Advanced Settings:

SETTING	SUPPORTED?
Operating time and Total time control variables	**Yes**
User-defined control variable	**Yes**
Number of Actions control variable	**Yes**
User-defined / Uncertain initial control variable value	**Yes**
Acceleration	**Yes**
Repair to Age	**Yes**

Specified Value Exceeded Failure Mode

The Specified value exceeded failure mode monitors a control variable and determines when its value exceeds a specified value.

The specified value exceeded failure mode supports initial ages, repair to age, and acceleration features.

Parameters:

PARAMETER	FULLY DYNAMIC	UPDATED WHEN REPAIRED/REPLACED	STATIC
Specified value	✓		

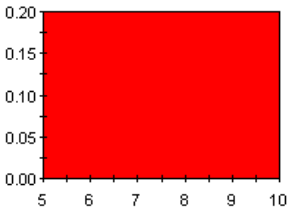
Advanced Settings:

SETTING	SUPPORTED?
Operating time and Total time control variables	**Yes**
User-defined control variable	**Yes**
Number of Actions control variable	**Yes**
User-defined / Uncertain initial control variable value	**Yes**
Acceleration	**Yes**
Repair to Age	**Yes**

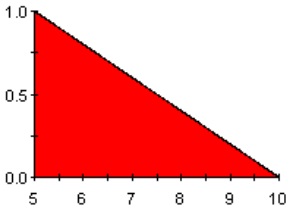
Uniform Failure Mode

The Uniform failure mode accepts two parameters – an upper and a lower bound on the control variable value at the time of failure.

The PDF of the uniform distribution has the following shape and equation:



$$f(x)=\begin{cases}\dfrac{1}{b-a} & for\ \ a\le x\le b\\ 0 & otherwise\end{cases}$$



$$R(x)=\begin{cases}1 & for\quad x\le a\\ 1-\dfrac{x-a}{b-a} & for\quad a\le x\le b\\ 0 & for\quad x>b\end{cases}$$

The Uniform distribution supports initial value options, repair to age and acceleration features. Both parameters can be dynamic, but the value of the control variable at failure is only recalculated when the component is placed in service, replaced, or the failure mode repaired.

Parameters:

PARAMETER	FULLY DYNAMIC	UPDATED WHEN REPAIRED/REPLACED	STATIC
Lower Bound		✓	
Upper Bound		✓	

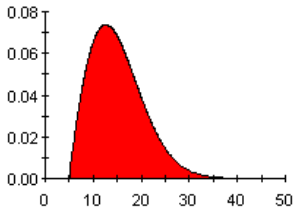
Advanced Settings:

SETTING	SUPPORTED	NOT SUPPORTED
Operating time and Total time control variables	✓	
User-defined control variable	✓	
Number of Actions control variable	✓	
User-defined / Uncertain initial control variable value	✓	
Acceleration	✓	
Repair to Age	✓	

Weibull Failure Mode

GoldSim supports two forms of the Weibull failure distribution. You can either specify the characteristic life and slope factor or the mean life and slope factor.

The PDF of the Weibull distribution has the following shape:



$$f(x) = \left(\frac{\alpha}{\beta}\right)\left(\frac{x}{\beta}\right)^{(\alpha-1)} e^{-\left(\frac{x}{\beta}\right)^{\alpha}}$$

where α is the shape parameter and β is the characteristic life.

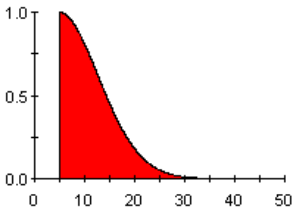
In GoldSim, you can directly specify the characteristic life and shape parameter using the Weibull - Characteristic life and slope factor failure mode. Alternatively specify the mean life of the component and the slope factor (using the Weibull – Mean life and slope factor failure mode). If you use this failure mode, GoldSim determines the characteristic life of the component using the formula for the mean value of the Weibull distribution, which is:

$$\mu = \beta\ \Gamma\left(1+\frac{1}{\alpha}\right)$$

This can be rearranged to solve for the characteristic life:

$$\beta = \frac{\mu}{\Gamma\left(1+\frac{1}{\alpha}\right)}$$

Regardless of the parameters used to quantify the distribution, the reliability equation for the Weibull distribution has the following shape and equation:



$$R(x) = e^{-\left(\frac{x}{\beta}\right)^{\alpha}}$$

In some cases, a Weibull distribution will actually be described with a third "location parameter" representing the time of the onset of the distribution. You can represent this in GoldSim by defining an input to the mode's acceleration factor that is zero until the time reaches the location parameter:

if (local.FMCV < LocationParameter then 0 else 1)

The Weibull distribution supports initial value options, repair to age and acceleration features. Both parameters can be dynamic, but the value of the control variable at failure is only recalculated when the component is placed in service, replaced, or the failure mode repaired.

Parameters:

PARAMETER	FULLY DYNAMIC	UPDATED WHEN REPAIRED/REPLACED	STATIC
Characteristic or Mean Life		✓	
Slope Factor		✓	

Advanced Settings:

SETTING	SUPPORTED?
Operating time and Total time control variables	**Yes**
User-defined control variable	**Yes**
Number of Actions control variable	**Yes**
User-defined / Uncertain initial control variable value	**Yes**
Acceleration	**Yes**
Repair to Age	**Yes**

Details of Failure Modes Available Only to the Action Element

Demand>Capacity Failure Mode

The Demand>Capacity failure mode is similar to an Event-triggered failure, but it is triggered when the component's action is triggered, and instead of a Probability of failure, it compares a demand with the component's capacity. If

demand exceeds capacity at the time the action is triggered, the component will fail.

The Demand>Capacity failure mode does not support control variables – it must be explicitly triggered by the element's Action. At that time, the expression or values specified for the Demand (typically a link to an outside element) and the Capacity are evaluated. If Demand exceeds Capacity, the failure mode will occur. It is important to note that the Demand and Capacity values are only evaluated at the time the action is triggered.

Because the failure mode does not use failure mode control variables, initial control variable values, acceleration and repair to age features are not supported.

Parameters:

PARAMETER	FULLY DYNAMIC	UPDATED WHEN REPAIRED/REPLACED	STATIC
Demand	✓		
Capacity	✓		

Advanced Settings:

SETTING	SUPPORTED?
Operating time and Total time control variables	**No**
User-defined control variable	**No**
Number of Actions control variable	**No**
User-defined / Uncertain initial control variable value	**No**
Acceleration	**No**
Repair to Age	**No**

Unreliable Failure Mode

The Unreliable failure mode is similar to an Event-triggered failure, but it is triggered when the component's action is triggered, and instead of a Probability of failure, it accepts a "Reliability" argument.

The Unreliable failure mode is quite different from the other failure modes in that it does not cause a failure which needs to be repaired. It models situations where a component might fail to operate correctly, but does not physically fail. A common use for this failure mode is modeling human interaction with a system (e.g., correct reaction to a system condition or problem). When an action occurs, GoldSim uses the Reliability value to determine whether or not the action is completed successfully. GoldSim first checks that the component's requirements are met and that none of its other failure modes or maintenance activities have occurred. If this is the case, it determines, using the Unreliable probability, if the action is successful. If the action is successful, an ActionOK event is issued. If it is unsuccessful, an ActionFailed event is issued, but the component does not fail as a result.

Because the Unreliable failure mode does not cause failure of the element, it cannot be repaired. In addition, since the failure mode does not use failure mode control variables, initial control variable values, acceleration and repair to age features are not supported.

Parameters:

PARAMETER	FULLY DYNAMIC	UPDATED WHEN REPAIRED/REPLACED	STATIC
Unreliable	✓		

Advanced Settings:

SETTING	SUPPORTED
Operating time and Total time control variables	**No**
User-defined control variable	**No**
Number of Actions control variable	**No**
User-defined / Uncertain initial control variable value	**No**
Acceleration	**No**
Repair to Age	**No**

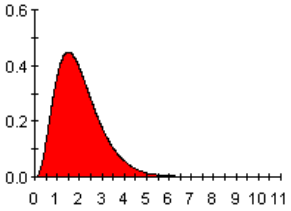
Details of the Repair Time Distributions

Determining When a Repair is Completed

When a failure mode that is repaired occurs, GoldSim samples the repair time distribution specified (repairs can be affected according to a Gamma, Lognormal or Exponential distribution) and schedules an event in the model's time queue for the anticipated time of repair. This ensures that regardless of the planned time steps, the repair will occur at the scheduled time.

Gamma Distribution

For repairs, the Gamma distribution is specified using a mean delay time until repair (μ) and a standard deviation (σ). The Gamma distribution has the following shape and equation:

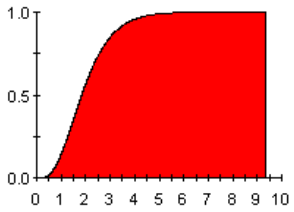


$$f(x) = \frac{\lambda(\lambda x)^{k-1} e^{-\lambda x}}{\Gamma(k)}$$

where:

$$k = \frac{\mu^2}{\sigma^2} \text{ and } \lambda = \frac{\mu}{\sigma^2}$$

The cumulative distribution function of the Gamma distribution has the following shape and equation:

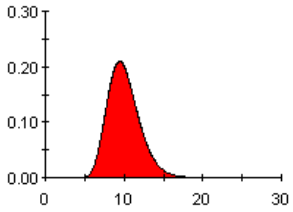


$$F(x) = \frac{\Gamma(k, \lambda x)}{\Gamma(k)}$$

The Gamma distribution is very flexible, as it equals an exponential distribution when $\mu = \sigma$, is similar to a lognormal distribution when $\mu > \sigma$, and approaches a normal when μ is small. This means the Gamma distribution can be used to model situations where the shape of the distribution of repair time values can change over the course of the simulation.

Lognormal Distribution

For repairs, the Lognormal distribution is specified using a mean delay time until repair and a standard deviation. The Lognormal distribution has the following shape and equation:



$$f(x) = \frac{1}{\sigma\ x\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\ln x - \mu}{\sigma}\right)^2}$$

where μ and σ are the mean and standard deviation of the logarithm of failure times.

In the Lognormal repair delay distribution μ and σ are not specified directly. The distribution accepts a true mean delay time until repaired and a true standard deviation.

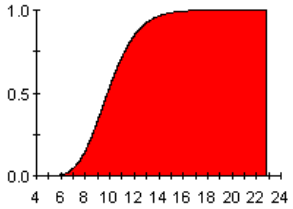
The true mean delay time until repaired is actually equal to:

$Mean = e^{\frac{\mu+\sigma^2}{2}}$ and the true standard deviation is equal to:

$$SD = \sqrt{(e^{\sigma^2} - 1)\ e^{2\mu+\sigma^2}}$$

GoldSim uses the above equations to solve for μ and σ.

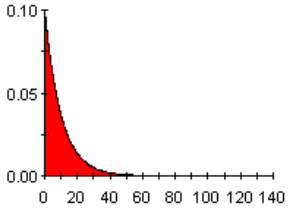
The cumulative distribution function of the Gamma distribution has the following shape and equation:



$$F(x) = \frac{1}{2}\left(1 + erf\left(\frac{\ln(x) - \mu}{\sigma\sqrt{2}}\right)\right)$$

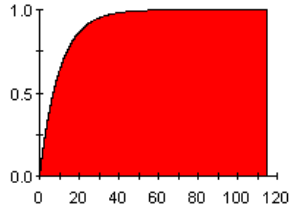
Exponential Distribution

For repairs, the Exponential distribution is specified using a mean delay time until repair (λ). The Exponential distribution has the following shape and equation:



$$f(x) = \frac{1}{\mu} e^{\frac{-x}{\mu}}$$

The cumulative distribution function of Exponential/Poisson distribution has the following shape and equation:



$$F(x) = 1 - e^{-\frac{x}{\mu}}$$

Computing Failure Distributions

The **Failure Times** button in the **Results** tab of a reliability element allows you to view a distribution of the time to failure (from all modes) for the component:

The times to failure represent the time intervals between when the component starts operating to when it fails (i.e., the time to failure). Of course, if repairs are taking place, a single realization of the system can have multiple time intervals between failures. The distribution that is displayed incorporates all intervals for all realizations.

Ideally, a simulation will have produced a large number of failures and a well-defined distribution will have been developed by GoldSim. However, for each realization where the component was unfailed at the end of the realization there is a potentially useful piece of information: a *censored* sample of the failure time

of the form "Failure time > X" , where X represents the time that the element had been operating as of the end of the realization. Sometimes these censored data are important, and as a result the following approach is used to incorporate them into the displayed distribution of failure times.

The distribution function is built up progressively by adding failures to a number of unequally-distributed time 'bins'. The censored data are integrated during this process, as follows:

- During the simulation, GoldSim records two data items for the start point of each bin: 1) the number of items with actual recorded failure times that were greater than the start bin; and 2) the number of censored failure times that were greater than the bin start time (but not greater than the next bin's start time).
- When preparing the failure time distributions, GoldSim smoothly interpolates a curve based on the binned results. Censored failure times are assumed to have a uniform distribution of their actual failure time spread over the balance of the unfailed population.
- If there are additional unallocated censored failure times that go beyond the last actually recorded failure (i.e., cases where the component never failed during an entire realization), GoldSim generates an assumed tail for the distribution, based on the statistics of the distribution so far. By default the distribution is assumed to be Weibull, but if the failures so far did not have a well-defined distribution (if three or less bins had results), the tail is assumed to be exponential.

Determining the Root Causes of Unmet Internal or External Requirements

Whenever a component fails due to unmet internal or external requirements the unique state of that component's logic tree is recorded, along with those of any other components that it relies on. These states can be analyzed to provide guidance as to which reliability elements responsible for the failure of the component currently being analyzed.

Note: If a component has both unmet internal and external requirements, only the state of its internal requirements tree is recorded.

When the **Root Cause** option is selected in the Causal Analysis dialog, GoldSim evaluates all of the internal and external trees that resulted in failure of the component and allocates responsibility to the elements that initiated the failure.

GoldSim assumes that the total responsibility of all components implicated in the failure must sum to 100%. GoldSim first checks the status of the top-level Internal Requirements node. If Internal Requirements prevent the element from operating, GoldSim deems the cause of failure to be internal to the element and assigns responsibility accordingly. It's important to note that unmet Internal Requirements are assumed to be causal even if the element's External Requirements are also unmet.

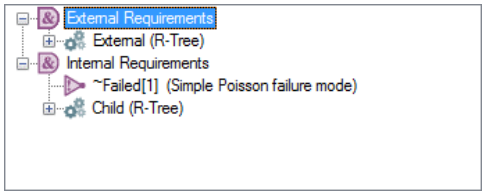
Working from the top level gate node, GoldSim divides responsibility for unmet requirements through the logic-tree to the node level.

Note: RL elements referenced in an Internal Requirements tree that are not operating due to the status of a parent are treated as operational when assigning responsibility for unmet requirements.

If an unmet requirement of a gate-note is a Condition or a Not node, the responsibility is assigned to the element where that node is specified. If an unmet requirement is an RLor Not RL Component node, the referenced RL element evaluates its requirements tree to try and determine if any lower-level elements were causal. This process continues until it identifies all the Causal Elements of the tree (these are RL elements that have failed due to a Condition or Not reference, are turned off, or are undergoing preventive maintenance). Once all Causal Elements have been identified, GoldSim multiplies the responsibility assigned to each reliability element by the time in that state or the number of times the state occurred (depending on whether Root Causes are being analyzed by Time in State or Occurrence Count) and assigns the responsibility to the corresponding element.

This process is repeated for all internal and external requirements tree states that resulted in failure of the element. The Time in State/Occurrence Count values for referenced elements are summed over all unique state trees, and the display is ordered from most to least responsible.

The methodology for assigning responsibility is probably best explained with examples. Consider the following requirements tree:



Scenario#1

Let's assume the External element is not operating, but that the element's Internal Requirements are met. In this scenario, the External element would be assigned responsibility for the failure of this element. The External element would then analyze the factors that prevented it from operating during this scenario (using the same procedure) and report back on the element or elements responsible for failure, and their culpability.

Scenario#2

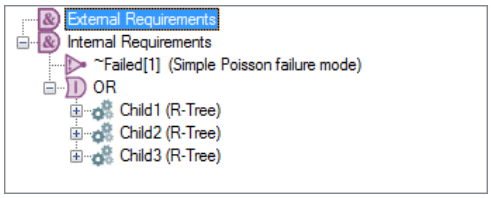
Again, let's assume the External element is not operating, but the Simple Poisson Failure Mode has occurred. Since there are unmet Internal Requirements this is assumed to be causal (even though the External element is not operating and the element's External Requirements are not met). Therefore, the ~Failed[1] reference receives 100% of the responsibility for the element's failure to operate. Since the responsibility for Condition and Not nodes lies with the referencing element, the referencing element will be declared causative and assigned 100% responsibility.

Scenario#3

Let's assume that the Simple Poisson Failure Mode has occurred and the Child element is not operating. Since both nodes of the Internal Requirements tree are false, each reference receives 50% of the responsibility for the element's

failure to operate. Since the responsibility for Condition and Not nodes lies with the referencing element, the referencing element will be declared causative and assigned 50% responsibility. The Child element will be assigned 50% responsibility and will then analyze its requirements tree to determine why it failed to operate, and report the lowest-level causative element(s) back to the referencing element.

Let's look at another scenario involving an OR-gate:



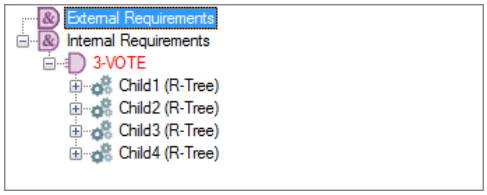
Scenario #4

Consider a situation where all three child elements are failed, but the Normal failure mode has not occurred. In this case, the Internal Requirements are unmet because of the false OR gate. As a result, 100% of the responsibility is assigned to that gate. Since all three Child elements must be failed for the gate to be false, each child is deemed to be causal and assigned 33% of the responsibility for the element's failure to operate. The Child elements will then analyze their requirements to determine if a lower-level element was causal.

Scenario #5

Consider a situation where all three child elements are failed, and the Normal failure mode has also occurred. In this case, the Internal Requirements are unmet because of the false OR gate and the false Not node. Both the OR gate and the Not node receive 50% responsibility for the element's failure to operate. Because Not and Condition nodes make the referencing element causal, the Not node means that 50% of the responsibility is assigned to the referencing element. In the case of the OR-gate, all three Child elements must be failed for the gate to be false. GoldSim assigns 16.67% (50%/3) of the responsibility for the element's failure to operate to those child elements. The Child elements will then analyze their requirements to determine if a lower-level element was causal.

The N-Vote can also distribute responsibility across referenced nodes:



Scenario #6

Consider a situation in the tree above where two of the four Child elements have failed. This will cause the 3-Vote gate to have a status of false. The two failed RL elements are each assigned 50% of the blame (responsibility is equally divided amongst false nodes of an N-vote gate). The implicated elements would then analyze their requirements to determine if a lower-level element was causal.

Calculation of Action Element Delays

Response of an Action Element Using a Delay Time

The response of an Action element using a delay time depends on its condition at the time the action is triggered.

When an element's action is triggered, the condition of the element is evaluated. If the element is not operating, an ActionFailed event is immediately issued. If the Action element is operating when its action is triggered, an ActionOK event will be emitted at the conclusion of the specified delay unless an Unreliable failure mode has been specified. However, progress through the delay is interrupted whenever the Action element is not operating. For example, if a component has a defined delay time of 30s, and the component stops operating 15s after its action is triggered, the corresponding ActionOK or ActionFailed event will be issued 15s after the component returns to service.

If an Action element is handling its Action internally, the condition of the parent element is not evaluated until its **OK** or **Failed** trigger is activated. This means that any child delays must occur before the parent can begin its delay.

If the element is operating when its Action is triggered, the delay time is computed based on a specified **Delay Type**. Three types of delays are available:

- Defined Delay Time.
- Defined Delay Time + Erlang Dispersion
- Defined Delay Time + Std. Deviation

The last two options allow the delay time to have a specified dispersion. That is, they allow you to simulate variability in the delay time after the element is triggered. This is equivalent to saying that there is a distribution of actual delay times around some mean, and whenever the element is triggered, the delay time for that event is sampled from a distribution.

Each option requires a specified **Delay Time** (or a **Mean Time** if dispersion is specified). The **Delay Time** and **Mean Time** must have dimensions of time and must be greater than zero.

Note: If the specified delay time is equal to zero, the event is emitted immediately (i.e., on the same update) without any delay. A negative delay time will result in a fatal error.

Action Event Delays without Dispersion

The simplest Action element delay is one which has a fixed delay time. In this case, after being triggered, the element "holds" the signal for a time period equal to the specified **Delay Time**, and after this delay, causes the Action element to emit an ActionOK or ActionFailed event (note that an ActionFailed event can only be emitted if an Unreliable failure mode has been specified).

Action Event Delays with Dispersion

In some cases, there may be variability in the time required to complete an Action. This is equivalent to saying that there is a distribution of actual delay times around the mean, and whenever the element is triggered, the delay time for that event is sampled from the distribution.

If we conceptualize the Delay as a conveyer belt for the triggered action, another way to view event dispersion is that as the action moves along the belt, it slips randomly forward or backward on the belt, with the amount of movement proportional to the degree of dispersion.

You can specify such dispersion in the delay time in two different ways:

- By specifying the dispersion in terms of an Erlang dispersion factor; or
- By specifying the dispersion in terms of a standard deviation.

If "Defined Delay Time + Erlang Dispersion" is selected, you must enter an **Erlang n-value**, which is a dimensionless value greater than or equal to 1. As n increases, the degree of dispersion decreases. As n goes to infinity, the dispersion goes to zero. The maximum amount of dispersion allowed is represented by n = 1, which corresponds to an exponential distribution.

If "Defined Delay Time + Std. Deviation" is selected, you must enter a **Std. Deviation**, which is a value with dimensions of time. The value must be greater than or equal to zero and less than or equal to the **Mean Time**. As the **Std. Deviation** decreases, the degree of dispersion decreases. When the **Std. Deviation** goes to zero, the dispersion goes to zero. The maximum amount of dispersion allowed is represented by **Std. Deviation** = **Delay Time**.

The Erlang n and the Std. Deviation are related by the following equation:

$$n = \left(\frac{\text{Delay Time}}{\text{Std. Deviation}} \right)^2$$

If the signal is dispersed, for every event, the Delay Time for that event is sampled from the following distribution:

$$f(t) = \frac{t^{n-1} e^{-t/\beta}}{\beta^n \Gamma(n)}$$

where:

n is the Erlang value (specified by the user);

β = D/n;

D is the mean delay time; and

Γ is the gamma function (*not* the Gamma distribution).

f(t) is the gamma probability distribution, which is equivalent to (and a generalization of) the Erlang distribution that is frequently used in simulation models.

The gamma distribution represents the time until the occurrence of n sequential Poisson-process events. Each event's random time is represented by an exponential distribution with mean D/n. The gamma distribution does not require n to be an integer. Note that for n=1, the distribution is exponential, and for increasing values of n it becomes less skewed, approaching normality for large n.

Note: When dispersion is specified for an Action Element's delay, each event which triggers the element is "assigned" an actual delay time by sampling from the distribution presented above. As a result, when Event Delays are dispersed, the events will not necessarily be "released" in the order that they were received.

Action Event Delays with Time-Variable Delay Times

In some cases, the delay time associated with the action will vary over time. If the delay time changes over the course of the simulation, GoldSim will treat the delay as if it is a conveyer belt. In particular, if the **Delay Time** or **Mean Time** is specified to become shorter (or longer) during the simulation (e.g., by defining it as a function of time), it can be imagined that the speed at which the

belt moves has simply been increased (or decreased), and all actions that are in the Delay at the time of the change start to move faster (or slower) by a common factor (the ratio of the old **Delay** or **Mean Time** to the new one).

For example, if the component's action was triggered at 10 s while the **Delay Time** was equal to 1000 s, and again at 15 s at which time the **Delay Time** was equal to 1 s, both signals would effectively be emitted at 16 s. That is, at the time that the **Delay Time** decreased, the first event would not have traversed a significant distance along the conveyer, and therefore it would be emitted from the conveyer just slightly in front of the second event.

Appendix B: Failure Mode Import Spreadsheet Format

> It is rare to find learned men who are clean, do not stink and have a sense of humour.
>
> Gottfried Wilhelm Leibniz

Appendix Overview

GoldSim provides the option to import failure mode information from a spreadsheet into Reliability elements. When the feature is enabled, GoldSim is able to import details of individual failure modes (including descriptions and repair information) from a user specified spreadsheet.

This appendix describes how the spreadsheet must be formatted in order to use it in this manner.

Note: The required spreadsheet format is illustrated in a template spreadsheet file FailureImport.xlsx found in the Reliability Examples folder in your GoldSim directory directory (accessed by selecting **File | Open Example...** from the main menu). Starting with this template file is by far the easiest way to create your own import spreadsheet

Required Spreadsheet Format

Failure mode information must start in Cell A3 of the worksheet containing the failure data. Row 1 and 2 can be used for header information or column headings. GoldSim expects the information in the following columns:

Column A: **Part ID (Required).** The Part ID is a unique identifier that Reliability elements use to retrieve data from the spreadsheet. Every element that is importing data from a spreadsheet must have a Part ID. In many cases, multiple elements will share the same Part ID (e.g., 5 pumps of the same type). This can be alphanumeric, and is not constrained by GoldSim's normal element naming conventions (leading numbers, numerical values and spaces are supported). It is not case sensitive.

Note: The spreadsheet rows do not need to be organized by Part ID and Failure Mode ID. GoldSim will automatically search for and locate all entries for a particular Part ID.

Column B**: Failure Mode ID (Required).** The Failure Mode ID will be assigned to the failure mode when it is imported. Like manually entered failure modes, the ID can be any number from 1 and 10, and all Failure Mode ID's for a particular Part ID must be unique.

Column C: **Failure Mode Type (Required).** The failure mode type is a short code indicating the type of the failure mode. These are:

Failure Mode Type	Short Code (not case sensitive)
Defective Component	DEFC
Erlang	ERLG
Exponential	EXP
Log Normal Geometric	LNG
Log Normal True	LNT
Normal	NORM
Specified Value Exceeded	SVE
Uniform	UNIF
Weibull, Characteristic Life & Slope	WBCL
Weibull, Mean Life & Slope	WBML
Unreliable (only supported by the Action reliability element)	UNR
Cumulative	CUM

Note: Two failure modes that can be entered manually (Event-triggered failure and Demand>Capacity) can not be imported from a spreadsheet, as their input parameters cannot be defined by simply entering constant values.

***Column D*: Description (Optional).** The contents of Column D are used to populate the description field for the corresponding failure mode.

***Columns E and F:* Failure Mode Parameter 1 and 2 (Required).** Data in these columns is used to populate the parameter fields for each of the failure modes. The definition of Parameter 1 and Parameter 2 differs for each type of failure mode, and correspond to the two parameter fields in the Failure Modes dialog, as summarized in the following table:

Failure Mode	Parameter 1	Parameter 2
Defective Component	Probability of defect	Rate of failure if defective
Erlang	Number of components	Rate of failure per component
Exponential	Rate of failure (hazard rate)	Unused
Log Normal Geometric	Geometric mean value at failure	Geometric standard deviation
Log Normal True	Mean value at failure	Standard deviation
Normal	Mean value at failure	Standard deviation
Specified Value Exceeded	Value at failure	
Uniform	Minimum value	Maximum value
Weibull, Characteristic Life & Slope	Characteristic life	Slope factor
Weibull, Mean Life & Slope	Mean life at failure	Slope factor
Unreliable (only supported by the Action reliability element)	Reliability	Unused
Cumulative	Number of time/survival pairs	Unused

***Column G:* Control Variable (Required).** This is a short code, corresponding with one of GoldSim's three built-in Failure Mode Control Variables (FMCV). The FMCV is the variable that is referenced by the failure mode to determine when failure occurs. (For those failure modes that are defined as distributions, the control variable represents the x-axis of a failure distribution plot.)

The control variables and short codes are as follows:

Failure Mode Control Variable	Short Code (not case sensitive)
Operating Time	O
Total Time	T
Number of Completed Actions (only valid for Action elements)	A

Note: When entering failure modes manually (as opposed to importing from a spreadsheet), you can also create a user-defined FMCV. However, user-defined FMCVs can not be defined and imported from a spreadsheet.

Column H: **Repair Type (Optional).** If a short code is specified in Column H, automatic repair will be enabled for that failure mode and the delay time distribution corresponding with the short code will be selected. The delay time distribution short codes are as follows:

Repair Time Distribution	Short Code (not case sensitive)
Gamma	RGAM
Lognormal (true mean and standard deviation)	RLNT
Exponential	REXP

Column I and J: **Repair Parameter 1 and 2 (Required if Repair Type specified).** Data in these columns is used to populate the parameter fields for each of the repair time distributions modes. The definition of Parameter 1 and Parameter 2 differs for each type of repair time distribution, and correspond to the two parameter fields in the Failure Modes dialog, as summarized in the following table:

Repair Time Distribution	Parameter 1	Parameter 2
Gamma	Mean delay time until repaired	Standard deviation
Lognormal (true mean and standard deviation)	Mean delay time until repaired	Standard deviation
Exponential	Mean delay time until repaired	Unused

Note: When entering repair time distributions manually (as opposed to importing from a spreadsheet), you can also specify a Resource requirement. However, Resource requirements for repair can not be defined and imported from a spreadsheet.

Columns K, K+1, … are used for time and survival fraction pairs for the Cumulative failure mode (e.g., K is time 1, K+1 is fraction 1; K+2 is time 2, K+3 is fraction 2, etc.). Note that the time values must start at 0, and must include a valid time unit (e.g., “10 day”).

Glossary of Terms

Availability

The probability that a component or system is performing its required function at any given time. Operational availability represents the probability that the component will be *operating* at any given time. Inherent availability is the probability that the component will be *operable* at any given time (a component that is turned off or has unmet external requirements, but is not failed is considered to be operable).

Causal Analysis

An analysis available within the Reliability Module that enables you to determine the root causes and system states that lead to system failures.

Discrete Change Signal

A discrete signal that contains information regarding the response to an event.

Discrete Event Signal

A discrete signal indicating that something (e.g., an accident, an earthquake, a failure) has occurred.

Discrete Signal

A special category of output that emits information discretely, rather than continuously.

Elements

The fundamental building blocks of a GoldSim model. Models are built by creating and manipulating elements, which represent the components of the system being modeled, data, and relationships between the data.

Failure Mode Control Variable

A variable that is referenced by each failure mode to determine when failure occurs. For those failure modes that are defined as distributions, the failure mode control variable represents the x-axis of a failure distribution plot. Each variable is defined with respect to a base variable (e.g., total time, operating time, mileage).

Fault-Tree

A logic tree in a Reliability Module element that that must evaluate to False in order for the component to operate.

FMCV

Failure Mode Control Variable.

Locally Available Properties

Special attributes of some elements in GoldSim. Locally available properties are similar to element outputs but do not appear as outputs of the element to which they belong. Rather, they are only visible in browsers. They derive their name from the fact that they may only be available, or they may take on

different values (i.e., be over-ridden), in “local” parts of your model (e.g., within particular elements or Containers).

Localized Container

A Container that has a separate scope such that elements inside the Container are treated as local (as opposed to global) variables.

Realization

A single model run within a Monte Carlo simulation. It represents one possible path the system could follow through time.

Reliability Modeling

Analyzing the ways that systems can fail (and be repaired) in order to increase their design life, and eliminate or reduce the likelihood of failures, downtime and safety risks. The output of these models typically consists of predictions of measures such as reliability (the probability that a component or system will perform its required function over a specified time period) and availability (the probability that a component or system is performing its required function at any given time).

Requirements-Tree

A logic tree in a Reliability Module element that that must evaluate to True in order for the component to operate.

Resource

Something that has a limited supply (e.g., spare parts, fuel, skilled personnel, money) and is required in order for elements of the modeled system to carry out certain actions.

Risk Analysis

An analysis that focuses on predicting the probability of those (presumably rare) failures that can lead to injury, loss of life, severe damage to the system, or perhaps damage to the surrounding environment. The output of a risk analysis generally consists of a probability of a particular unlikely, but high consequence outcome (e.g., catastrophic failure of the system), and identification of those events or components most likely to lead to that outcome.

Store

Stockpiles or places where a Resource (e.g., parts, personnel) is stored or located when not being used. Resource Stores can be thought of as having physical locations in the system you are modeling. The can be global or local (associated with a Container).

Index

G

I

K

L

M

N

O

P

R

S

T

U