A Dynamic Simulation Approach to Reliability Modeling and Risk Assessment Using GoldSim

White Paper

GoldSim TECHNOLOGY GROUP

Visit us at www.goldsim.com
Contact us at software@goldsim.com

Table of Contents

Introduction

Things fail. Some failures are simply inconveniences, while others have can have significant economic and societal impacts (e.g., resulting in loss of life).

Reliability modeling involves modeling the ways that systems can fail (and be repaired) in order to help determine how 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) that work together to carry out one or more functions. 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(s) over a specified time period) and ***availability*** (the probability that a component or system is performing its required function(s) at any given time). Reliability models are typically used to compare design alternatives on the basis of metrics such as throughput, warranty and/or maintenance costs.

For some systems, the analyst may be more concerned with (probabilistic) ***risk assessment*** than with reliability. Probabilistic risk assessment (PRA) was initially developed to analyze complex systems such as nuclear power plants and space missions. It 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 PRA, 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 assessment models are typically used to evaluate system safety and inform decisions regarding the allocation of resources (e.g., design or operational changes) to accident prevention.

Although reliability modeling and risk assessment share some common features (e.g., they both deal with failure of various components and systems), these two types of analyses traditionally use different types of approaches (since they are focused on different types of results). This document discusses how ***GoldSim***, a dynamic probabilistic simulation program, can be used for both types of analyses. Simulation-based approaches such as that used by GoldSim can make it possible to tackle complex reliability and risk assessment problems that cannot be easily or realistically addressed using traditional approaches.

For reliability modeling, the fundamental outputs produced by GoldSim consist of traditional reliability metrics (e.g., reliability and availability) for the overall system, and for individual components within that system. For risk assessment, GoldSim can be used to compute the probability of specific consequences (e.g., an accident leading to loss of life) to support risk management for the system. GoldSim also catalogs and analyzes failure scenarios, which allows for key sources of unreliability and risk to be identified (i.e., ***root cause analysis***).

However, the true power of GoldSim is that it can do more than compute only these kinds of reliability and risk management metrics. This is because GoldSim differs from the few existing simulation-based approaches to reliability and risk assessment in that it combines powerful features for representing the failure (and repair) of complex systems with the flexibility to represent the true dynamic complexity and evolution of the entire system. That is, GoldSim is first and foremost a powerful and extremely flexible general-purpose, probabilistic, dynamic simulator that has been used to simulate the behavior and

evolution of a wide variety of complex systems ranging from environmental systems (e.g., mines, watersheds, waste disposal sites) to engineered systems (e.g., processing facilities, machines, space missions) to business systems (e.g., companies, projects).

By combining these fundamental capabilities with the Reliability Module, a specialized extension for dynamically modeling the failure (and repair) of engineered components, GoldSim makes it possible to build "total system models" that can represent 1) ***evolving environmental conditions***; 2) the realistic, ***dynamic complexity of failure*** of components within the system (e.g., complex interdependencies, failure rates that respond to evolving environmental conditions); and 3) the actual ***consequences*** of failure (e.g., changes in throughput, costs, loss of life, and other measures of system performance).

Purpose and Outline

The purpose of this White Paper is to explain how GoldSim can be used for reliability modeling and probabilistic risk assessment. The document is longer than the typical White Paper, as the objective is not just to describe GoldSim in simplified, broad terms (i.e., "arm-waving" that provides very little insight), but instead to provide sufficient detail such that the reader can obtain a good understanding and overview of what the software can actually do (and how it does it). Because GoldSim is very powerful and flexible, doing so requires more than just a few pages (although this document contains lots of screen captures, so it is not as long as it might seem). Note, however, that the paper does not attempt to teach you how to actually use the software; it is intended to simply clearly explain its capabilities in simple language. Readers interested in learning more details are pointed to additional sources of information at the end of the paper.

In order to illustrate how GoldSim can be used for reliability analysis and probabilistic risk assessment, the document is organized as follows:

- How is GoldSim Different from Traditional Approaches? First we provide a very general overview of how GoldSim differs from traditional approaches to reliability modeling and probabilistic risk assessment.
- Basic GoldSim Concepts. In order to demonstrate how GoldSim can be used for reliability modeling and probabilistic risk assessment, it is first necessary to provide a brief overview of the basic concepts underlying simulation modeling in general, and more specifically, the GoldSim simulation framework.
- GoldSim's Approach to Reliability Modeling. After obtaining an understanding of basic GoldSim concepts, it is then possible to illustrate how GoldSim can be used for reliability modeling. This is done by showing a number of example models.
- GoldSim's Approach to Probabilistic Risk Assessment. This builds upon the previous section to illustrate how GoldSim can be used for probabilistic risk assessment. Several aerospace case studies are discussed to illustrate the key concepts.
- Summary. The document will conclude with a brief summary and a description of ways in which you can learn more about GoldSim.

How is GoldSim Different from Traditional Approaches?

When discussing how GoldSim differs from other approaches, it is useful to differentiate reliability modeling from probabilistic risk assessment. GoldSim can be used for both types of analyses. With traditional approaches, however, these two types of analyses use different types of tools (since they are focused on different types of results).

Traditional Approaches to Reliability Modeling

It is assumed that the reader is familiar with traditional reliability modeling approaches. Ebeling (2009) is a good introductory text that discusses these approaches.

Most traditional 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 traditional 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 kinds of 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 Exponential 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/Closed-Form Solutions. Reliability block diagrams can be used to formulate closed-form solutions when modeling many systems of components. Such models, however, are static, assume the system is in steady state, and do not account for the highly dynamic nature of many systems. Moreover, unless (simplistic) correction factors are used, the approach 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 and can be used to represent systems that cannot be handled using reliability block diagrams and closed-form solutions. 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, as we will discuss below, in some cases a more realistic reliability model may be required.

Traditional Approaches to Probabilistic Risk Assessment for Engineered Systems

Risk assessment is a very broad field, utilizing a variety of quantitative approaches. In the current context, however, we are primarily concerned with risk assessment 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 assessment 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 logic-based procedures (e.g., ***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 and/or have complex dependencies among failure processes may be difficult to realistically represent and/or may require a tremendous amount of preprocessing effort when using event tree/fault tree approaches.

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

Note: Stamatelatos et al. (2011) is the latest version of NASA's *Probabilistic Risk Assessment Procedures Guide for NASA Managers and Practitioners*. In addition to discussing traditional logic-based procedures in detail, it also briefly discusses simulation-based approaches, and in fact, presents an example using GoldSim. Mattenberger et al. (2015) provides a comparative analysis of a simulation-based approach to PRA (specifically using GoldSim) to traditional approaches (for crewed spacecraft missions).

The GoldSim Approach to Reliability Modeling and Risk Assessment

GoldSim is a general purpose dynamic, probabilistic (Monte Carlo) simulator. Dynamic simulation allows the analyst to develop a representation of the system, and then observe that system's predicted 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 (e.g., environmental).
- If some system properties are uncertain, the significance of those uncertainties can be determined.

In a dynamic, Monte Carlo simulation, the dynamic behavior of the system (e.g., evolving environment, various failures and repairs, system performance) is simulated many times. These multiple results (referred to as ***realizations*** of the system) can then be combined to provide not only a mean, but also a range on the performance of the system. In addition to the statistical results 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 readily 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 the external environment: Because GoldSim is a general purpose simulator, the environment in which the system operates can be readily modeled, and can affect and interact with the system.

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 operating rules. Components can be specified to turn on and off according to a fixed schedule and/or in response to external events. This allows accurate calculation of availability, and can also affect failures (since failures based on distributions can choose, among other things, to use total time or operating time as the control variable).

Model complex interdependencies: In addition to providing a logic-tree mechanism to define relationships (e.g., the power supply must be operating in order for the rest of the system to operate), 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. You can also easily model non-fatal failures (i.e., failure modes that only partially degrade the performance of a component).

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

In the remainder of this document, we will explain in more detail (using example models) how GoldSim can be used to represent such systems.

Basic GoldSim Concepts

Before describing how GoldSim can be used for reliability modeling and risk assessment, it is first necessary to provide a brief overview of the basic concepts underlying simulation modeling in general, and more specifically, the GoldSim simulation framework.

Simulation Concepts

GoldSim carries out ***dynamic, probabilistic simulations***. The term "simulation" is used in different ways by different people. 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 project, an organization, a facility, an ecosystem, a mission, a machine) 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 that can be quantitatively described using equations and/or rules can be simulated.

In a ***dynamic simulation***, the system changes and evolves with time (in response to both external and internal influences that the analyst specifically defines), and your objective in modeling such a system is to understand the way in which it is likely to evolve, predict (forecast) the future behavior of the system, and determine what you can do to influence that future behavior. That is, the purpose of a dynamic simulation is typically to predict the way in which the system will evolve and respond to its surroundings, so that you can identify any necessary changes that will help make the system perform the way that you want it to.

A ***probabilistic simulation*** recognizes that the controlling parameters, processes and events for any system you are trying to simulate may not be able to be predicted with certainty and/or may not be well understood, and it therefore attempts to represent this uncertainty explicitly and quantitatively. In this regard, there are two fundamental types of uncertainty that it is important to distinguish between and represent:

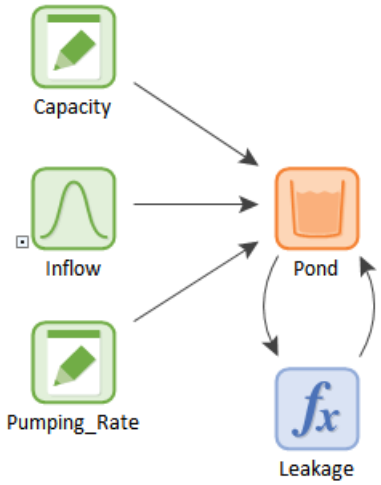
1) that due to inherent (temporal) randomness (e.g., a ***stochastic*** process); and
2) that due to ignorance or lack of knowledge.

Failures are a classic example of the first item: we may be able to describe failures statistically, but when the actual failures occur is inherently random. On the other hand, the second item reflects the fact that in some parts of our system, we may simply have a lack of knowledge regarding a particular variable (e.g., the strength of a material, or the properties of a soil).

GoldSim is able to represent both types of uncertainty. It does this by quantitatively representing the uncertainty in inputs (e.g., using distributions describing failure rates, the rates of other events, and the uncertainty in key variables). Uncertainty in inputs is propagated to the uncertainty in the outputs using ***Monte Carlo simulation***. In Monte Carlo simulation, the entire system is simulated a large number (e.g., 1000) of times. Each of these simulations is referred to as a ***realization*** of the system. For each realization, all of the parameters described by distributions are "sampled" (for distributions representing failure rates or other stochastic processes, multiple times). The system is then simulated through time such that the outputs of the system can be computed. This results in a large number of separate and independent results, each representing a possible "future" for the system (i.e., one possible path the system may follow through time). The results of the independent realizations are assembled into probability distributions of possible outcomes.

What is GoldSim?

Let's walk through a very simple example that illustrates these concepts. GoldSim is essentially a high-level programming language for building simulation models (but does not require you to be a computer programmer). It is highly-graphical and object-oriented, such that you create, document, and present models by creating and manipulating graphical objects representing the components of your system, processes, data and relationships between the data:



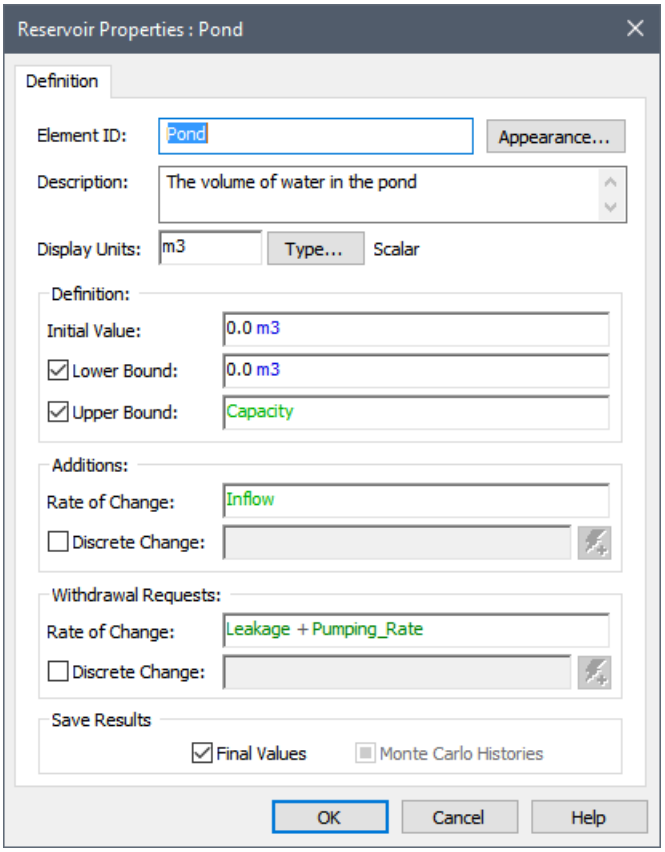
The simple model above has five objects: Capacity, Inflow, Pond, Leakage and Pumping_Rate. Each of these objects represents a feature (e.g., a pond), a parameter or property (the capacity of the pond), or a process or event (inflow and leakage from the pond). The objects representing features, parameters, processes, and events in GoldSim are called ***elements***. The purpose of this particular model is to predict the volume of water in the pond as a function of time, accounting for specified inflows and outflows.

> **Note**: This particular example intentionally does not model failures (e.g., which could affect the pumping rate); we will address that in subsequent sections. Here we will simply use this example to illustrate the fundamental concepts of dynamic, probabilistic simulation.

Elements are the fundamental building blocks of a GoldSim model, and each type has a particular symbol or graphical image by which it is represented on the screen. You give each element a unique name by which it is referenced. GoldSim provides a wide variety of elements (over 50), each of which serves a different purpose. Some of these elements simply provide a mechanism for the user to enter input data into the model (e.g., Capacity, Pumping_Rate). Other elements represent functions which operate on one or more inputs and produce one or more outputs (e.g., Leakage). Some elements represent uncertaint parameters or stochastic processes (e.g., Inflow). And other classes of elements are relatively complex and generate the internal dynamics of a model. In this simple model, the element Pond serves this purpose.

In particular, the element named Pond (referred to in GoldSim as a ***Reservoir***) integrates material flows over time. In this case, the material is water. The Reservoir element solves a time integral: it integrates the inflows and outflows, and by doing so in this case computes the volume of water in the pond at any time in the simulated future.

As pointed out, as a general rule, each type of element in GoldSim has one or more inputs and produces one or more outputs. Each element has a ***properties dialog*** where the inputs are entered. The properties dialog for the Reservoir element representing the Pond looks like this:



Note that when you link one element to another (e.g., by referencing another element in an input field as shown above), GoldSim automatically draws an arrow (referred to as an ***influence***) between the elements. The influence visually indicates the dependency of one element on another. In the example above, the influences indicate that:

- Pond is influenced by (i.e., is a function of) Capacity, Inflow, Leakage and Pumping_Rate.
- Leakage is influenced by Pond (which forms a ***feedback loop*** between these two elements).

One of the more unique and powerful features of GoldSim is that the program is dimensionally aware. GoldSim has an extensive internal database of units and conversion factors. You can enter data and display results in any units. For example, you could add meters and feet in an equation, and GoldSim would internally carry out the conversion. Note, however, that if you tried to add meters and hours, GoldSim would issue a warning message and prevent you from doing so.

When elements are created, you must specify their output dimensions. When elements are linked, GoldSim ensures dimensional consistency and carries out all of the unit conversions internally. In this particular example, the Pond has **Display Units** of a volume (m3). As a result, GoldSim expects the **Upper Bound** (Capacity) to have dimensions of a volume, and the **Rate of Change** (Inflow, Leakage and Pumping_Rate) to have dimensions of volume per time. If they did not, GoldSim would display an error.

Running a Model

Systems that are changing with time are described mathematically using differential equations. In the simple example shown above, we have only a single variable (the volume), so this can be described using an ordinary differential equation as follows:

$$\frac{dV}{dt} = \text{Inflow - Outflow}$$

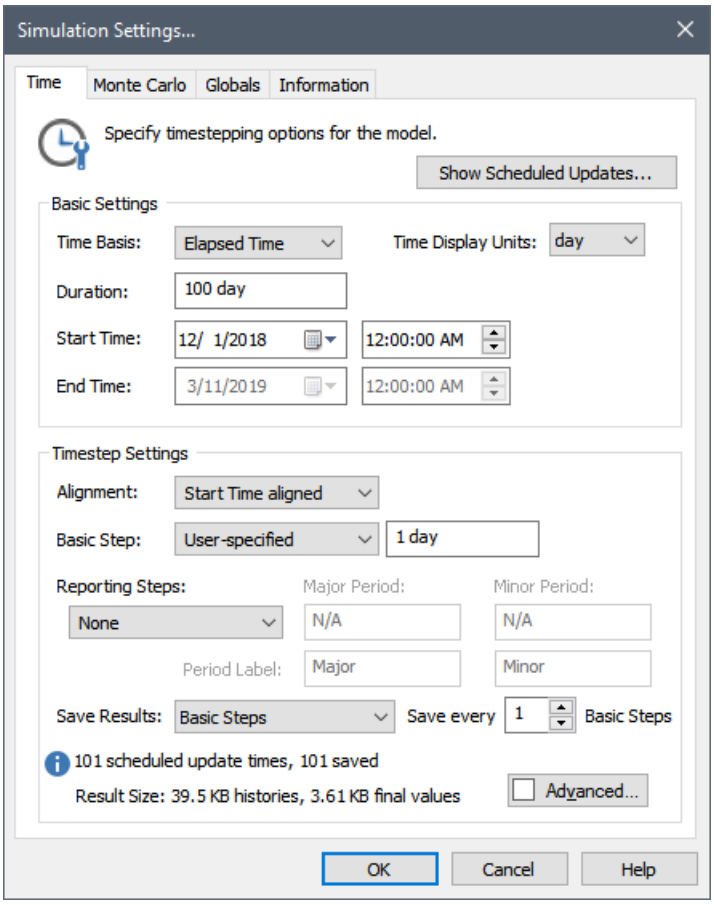
To write this in terms of the volume, we take the integral:

$$V(\tau) = V(0) + \int_0^\tau (\text{Inflow - Outflow})\, dt$$

To solve for the volume as a function of time, we need to solve this integral. In simple systems (e.g., if the flows were constant), we can solve this analytically. For almost any real system that you would be interested in modeling, however, an analytical solution is not available. Therefore, a dynamic simulator like GoldSim must solve such equations numerically (by computing an approximate solution). This is what the Reservoir element does.

To solve this (or any) integral numerically, it is necessary to discretize time into discrete intervals referred to as ***timesteps***. GoldSim then "steps through time" by carrying out calculations every timestep, with the values at the current timestep computed as a function of the values at the previous timestep.

Hence, in order to dynamically simulate a system in GoldSim, you must specify the duration of the simulation (e.g., 1 year) and the length of the timestep (e.g., 1 day). GoldSim provides a flexible dialog for specifying how it steps through time:



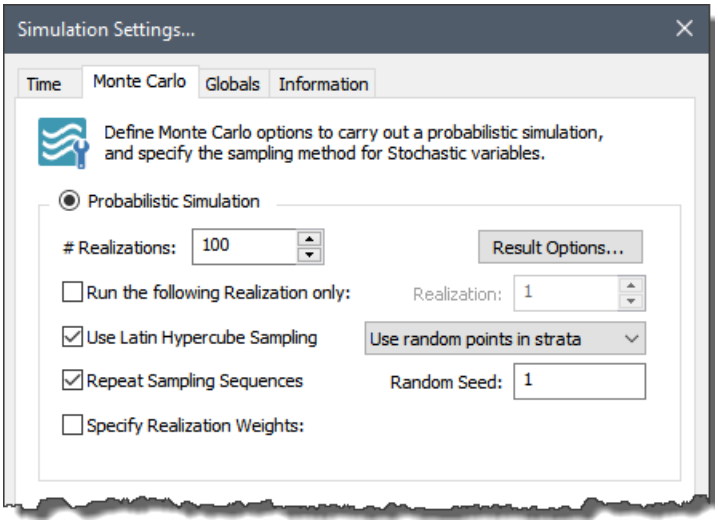
There are two ways to carry out a dynamic simulation in GoldSim (specified by selecting the **Time Basis** in this dialog):

- In an "Elapsed Time" simulation, you specify a simulation **Duration**. The simulation is then tracked in terms of the elapsed time since the simulation began.
- In a "Calendar Time" simulation, you enter a **Start Time** and an **End Time**, and the simulation is tracked in terms of the calendar date/time (i.e., GoldSim tracks things like what hour of the day, day of the week and month it is during the simulation, and you can explicitly refer to these in the simulation).

If the simulation you want to run is very short (e.g., minutes or hours) or very long (e.g., hundreds of years), in most cases an Elapsed Time simulation would be appropriate. However, when your simulation duration is between these two extremes, it is quite possible that you will want to run a Calendar Time simulation. This is because some inputs, or the behavior of the system itself, might depend on the time of day or the date (i.e., parameters may have diurnal and/or seasonal patterns), and hence you will want to specifically track and reference this information in your model .

Note: To ensure that the numerical approximations in a dynamic simulation are accurate, a sufficiently small timestep must be used. The appropriate timestep length is a function of how rapidly the system represented by the model is changing: the more rapidly it is changing, the shorter the timestep required to accurately model the system. It is important to note, however, that the actual timestep length in a simulation is not necessarily constant, and in fact, when simulating events (such as failures), GoldSim automatically inserts timesteps in order to accurately simulate them. That is, if an event occurs at a particular time, GoldSim can interrupt the simulation and update the model. For example, consider a failure that occurs 12.54 days into the simulation, and is repaired 0.72 days later. The simulation would be updated (i.e., a timestep would be inserted) at 12.54 days to reflect the failure, and would subsequently be updated at 13.26 days to reflect the repair. This allows GoldSim to model such systems without an inordinate level of computational effort.

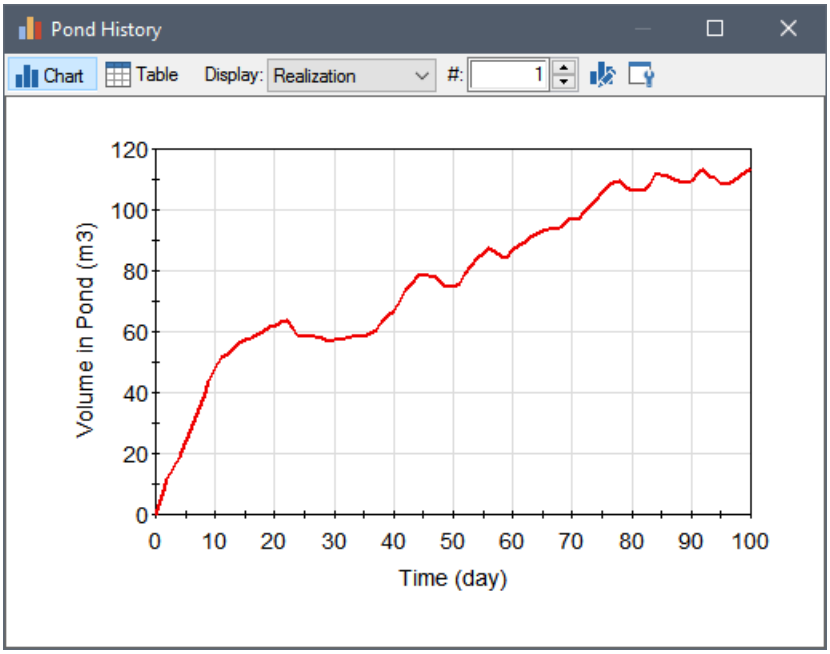
If you are running a probabilistic simulation, you must also specify how many realizations of the model you would like to run:



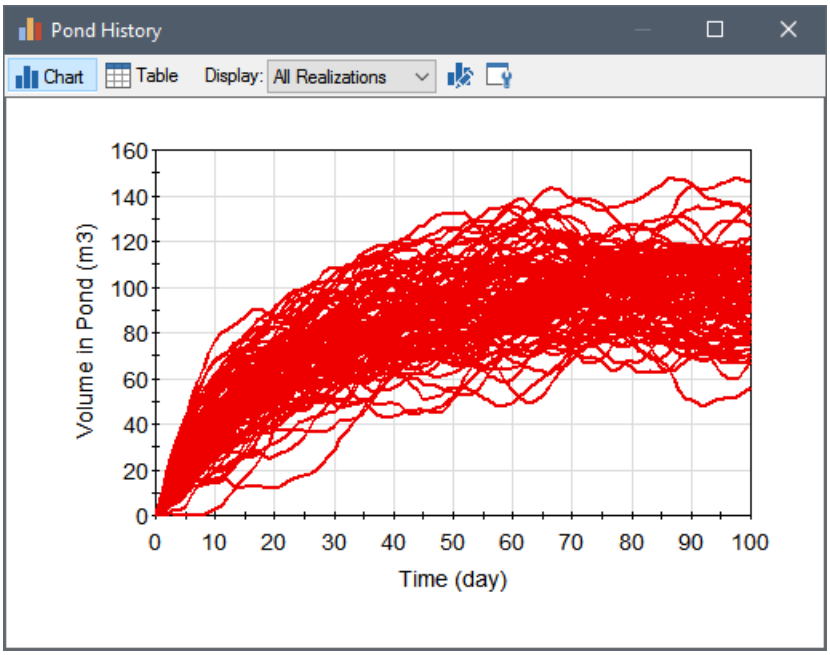
Displaying Basic Simulation Results

After running a model, GoldSim can generate and display different types of results, in either graphical or tabular form. The most common results viewed are ***time history results*** and ***distribution results***.

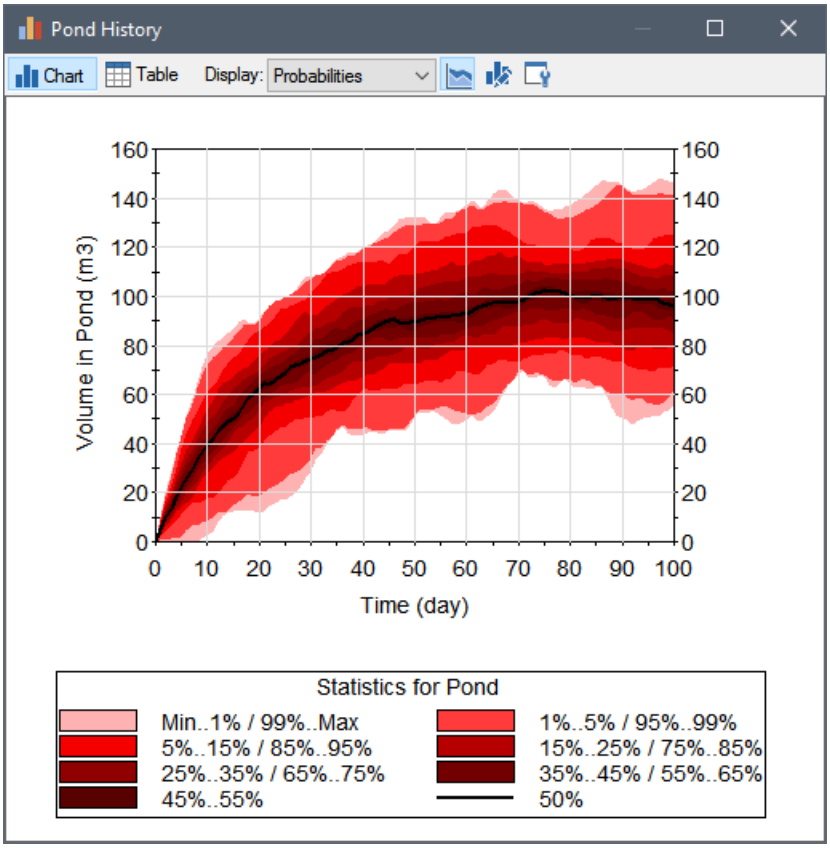
A time history result simply shows how a model output is predicted to change with time. As such, it is the fundamental type of result produced by a dynamic simulation model. The x-axis is elapsed time (or date/time) and the y-axis is the value of the output. Here is an example of a simple time history for our pond model:



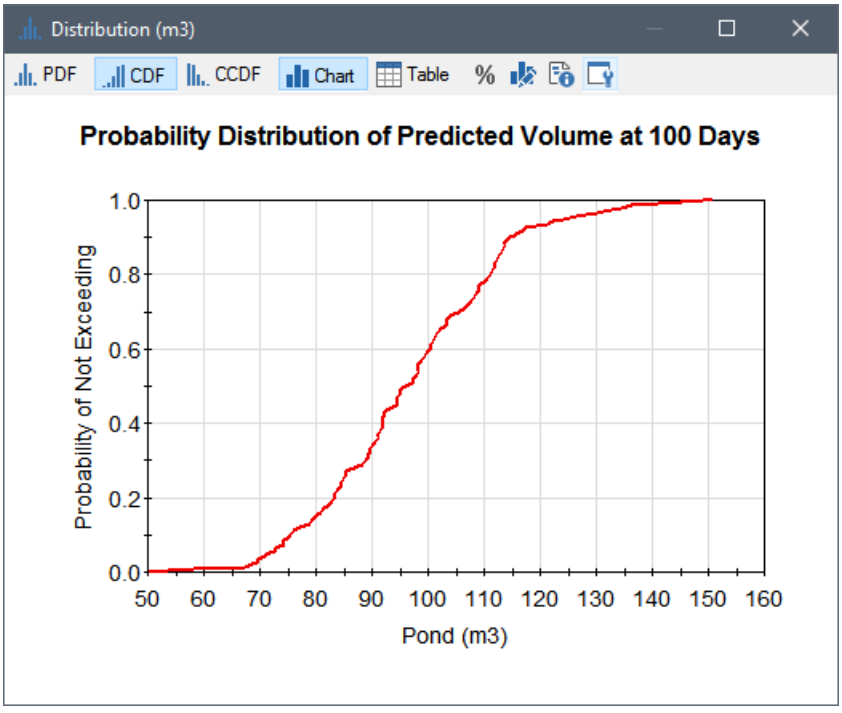
This is the plot of a single realization. In a probabilistic model, we run multiple realizations (each representing a possible future). Here we show 100 realizations:



A more useful way to display multiple realizations (i.e., a probabilistic time history result) is to display it in the form of percentile bands:



A ***distribution result*** shows a probability distribution of an output at a specific point in time (e.g., the end of the simulation):



Here we are displaying the result in terms of a ***cumulative distribution function*** (CDF). The y-axis shows the probability of not exceeding the value on the x-axis. So in this example, if we look at an x-axis value of 100 m3, we see there is about a 60% chance that volume at the end of the simulation will not exceed that value (and hence a 40% chance that it will exceed that value).

Modeling Events

Use of GoldSim for modeling reliability and risk assessment requires a basic understanding of one set of powerful features in GoldSim: discrete event modeling.

When things move through or change within a system, the dynamics can be conceptualized in two different ways: ***continuously*** or ***discretely***. Things that move continuously can be thought of as flowing. An example of this is the movement of water. Other things move or happen discretely or instantaneously (e.g., such that they must be tracked individually). Examples of this include financial transactions, the movement of items through a factory, and, of course, failures and repairs.

The example we discussed above dealt only with continuous dynamics (the flow of water). It is important to understand, however, that GoldSim provides powerful capabilities for representing discrete dynamics as well. In fact, most real-world systems are best described using a combination of continuous and discrete dynamics (i.e., hybrid systems). And because failures and repairs are discrete events, the ability of GoldSim to properly handle these is critical.

GoldSim allows you to represent “instantaneous” changes to a model by providing a mechanism for a model to generate and respond to events. This is accomplished by 1) providing the ability to generate events in a number of different ways, and 2) allowing such events to instantaneously ***trigger*** various elements to take a particular action (e.g., instantaneously change their value).

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

1. The event occurs when a specified condition (e.g., X > Y) becomes true or false;
2. The event occurs when a specified output in the model changes;
3. The event occurs at a specified calendar or elapsed time;
4. 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"); or
5. The event occurs (failures and repairs) based on specified failure models, conditions and interdependencies.

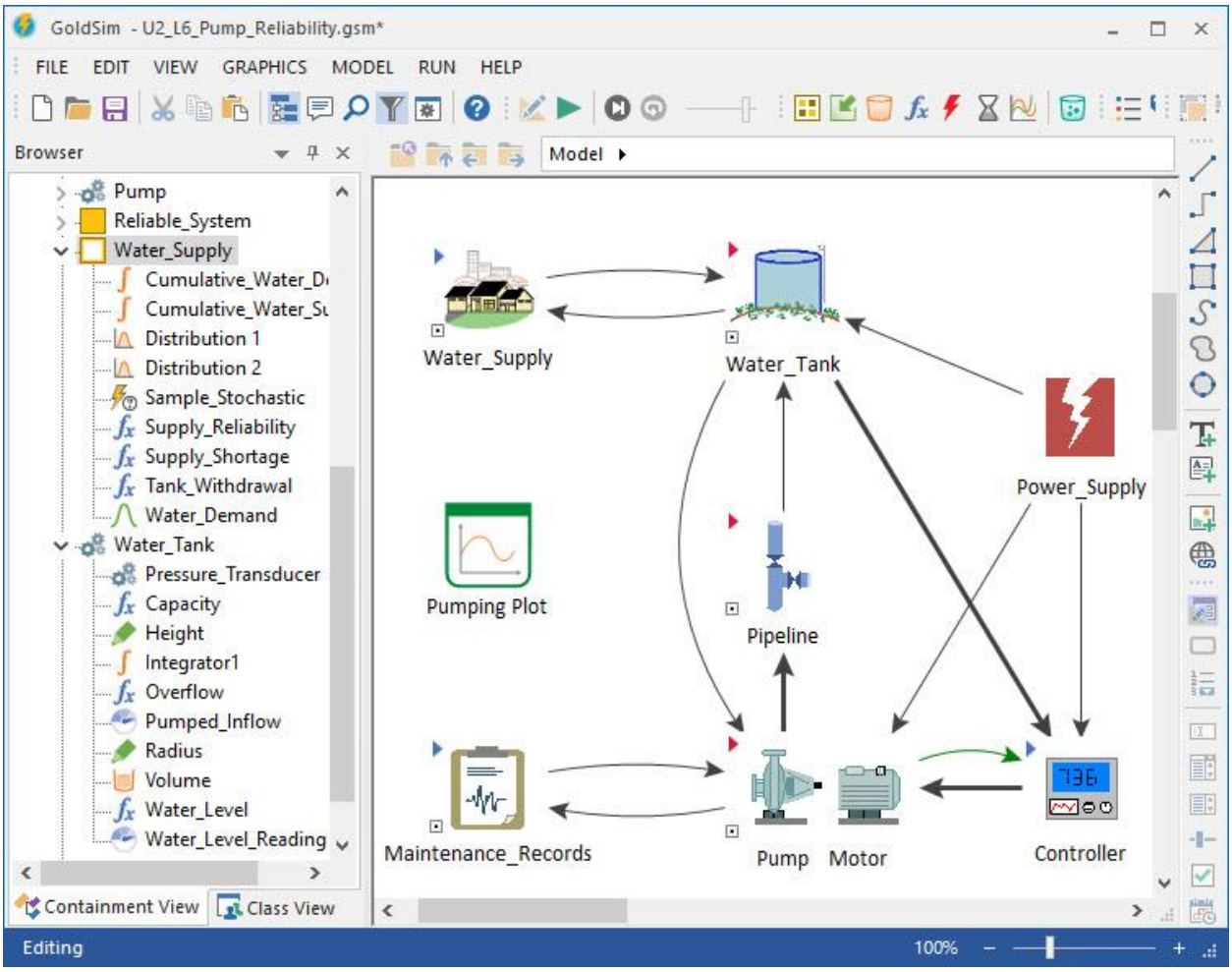
Once an event is generated, a variety of GoldSim elements can be triggered by the event, with each element responding to the event (taking a particular action) in a different manner. The ability to superimpose the effects of events (such as failures) on continuously varying systems (in order, for example, to model consequences) is one of the most powerful features of GoldSim.

Building Large, Hierarchical Models

Although some GoldSim models are very simple (such as the simple example above), consisting of a small number of elements, complex GoldSim models can have hundreds or thousands of elements. In order to manage, organize and view such a model it is useful (in fact, essential) to group the elements into ***Containers***. A Container is simply a collection of elements.

A Container can be thought of as a "box" into which other elements have been placed. In a sense, it is like a directory folder on your computer. The elements inside the Container can be thought of as a "sub-system" of your model. Containers can be placed inside other Containers, and any number of levels of containment can be created. This ability to organize model elements into a hierarchy provides a powerful tool for creating "top-down" models, in which the level of detail increases as you "drill down" into the containment hierarchy.

The example below shows a system that has been divided into a number of distinct sub-systems:



All of the elements with a small triangle in their upper left-hand corner are Containers. Clicking on the triangle allows you to drill down into (i.e., enter) that Container to see more details. The hierarchy and contents of the Containers are shown in the tree structure on the left side of the screen. The elements inside a particular Container are shown on the right side of the screen.

> **Note**: As we will see below when we discuss how GoldSim models failures in reliability and risk assessment models, Containers play a critical role in representing ***systems*** of components.

The ability to create sub-systems using Containers provides a powerful capability: the reuse of sub-systems. A user can create a complex sub-system, and then document and save it, such that a

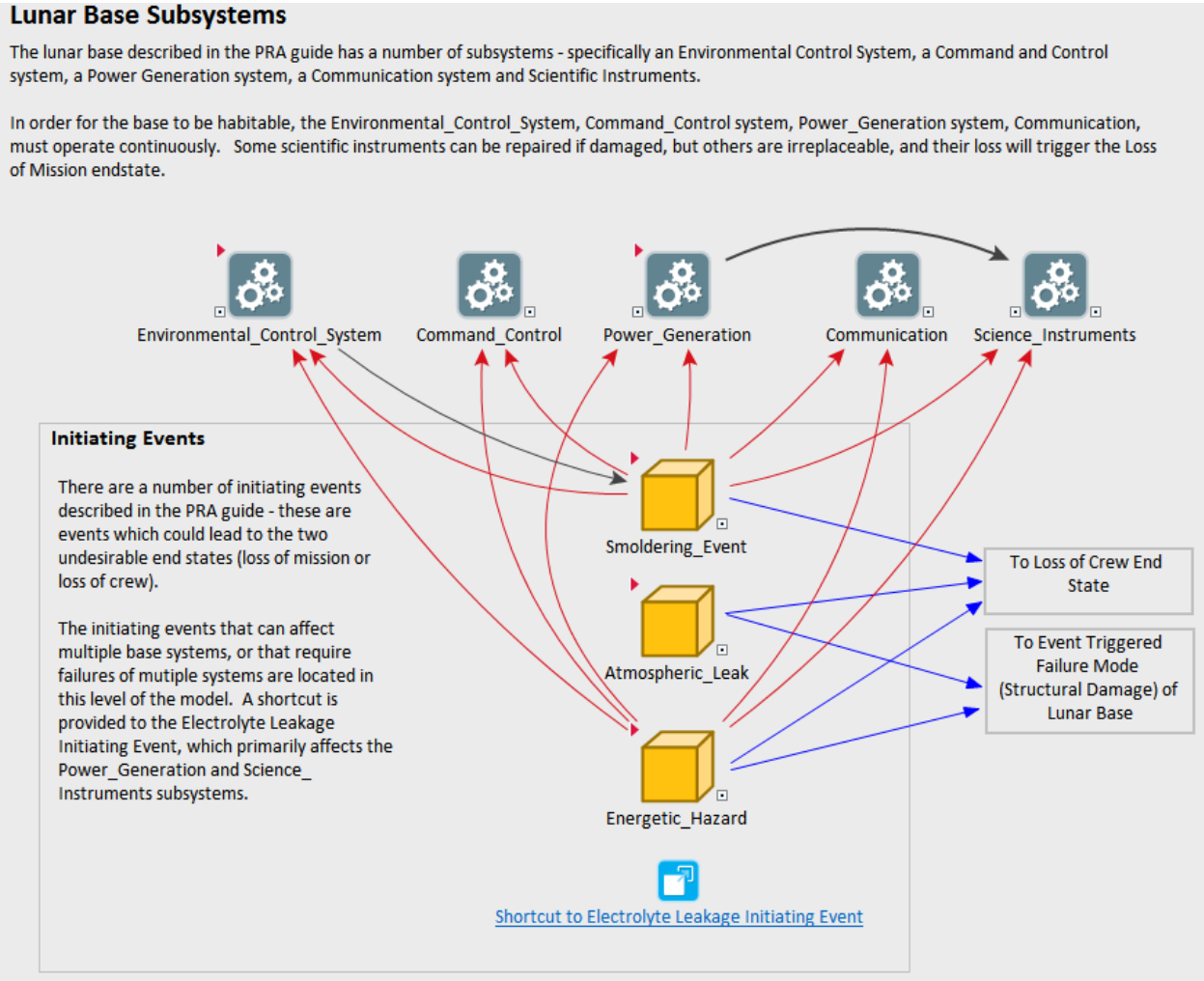
subsequent user could simply drop the sub-system into a new model. This facilitates the creation of a library of documented and verified sub-systems. Such a library can be used to quickly and efficiently build complex models.

Building Transparent, Well-Documented Models

A key feature of any modeling tool is how well it allows you to document and explain your model. Properly documenting your model is critical for three important reasons:

- Many models have a long lifetime. As a result, you will often need to revisit and make modifications to a model many months (or years) after you last used it. If the model is not well documented, you will need to waste time coming back up to speed with the model in order to understand it well enough to use the model and make any modifications that are necessary.
- Many models are either built by multiple people, or pass from one person to another over time. In order for others who need to work on the model to do so effectively, it must be well documented.
- Most models that an analyst builds are actually built for someone else (e.g., a manager, a client, a regulator, some other stakeholder). Although it may not be necessary for them to understand all of the technical details of a model in order to use it, in most cases it is necessary for them to understand the basics of what the model is doing. A model which cannot be easily understood is a model that will not be used or believed. A well-documented model is more likely to be used by the stakeholders for whom it was built.

As a result, GoldSim was specifically designed to allow you to effectively document, explain and present your model directly inside of the model itself. You can add graphics, explanatory text, notes and hyperlinks to your model:



GoldSim's powerful documentation and presentation abilities, coupled with the ability to create hierarchical, top-down models, allows you to effectively describe and explain your model at different (and appropriate) levels of detail to different audiences.

Summary

The purpose of this section was to provide a very short introduction to the basic concepts upon which GoldSim is based. This overview, although very brief, provides sufficient background information for us now to describe in some detail how GoldSim can be used for reliability modeling and risk assessment.

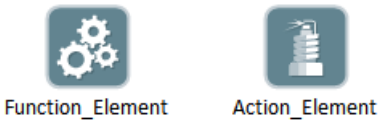
GoldSim's Approach to Reliability Modeling

In this section, we will describe in some detail how GoldSim can be used for reliability modeling. We will start by describing some very simple problems, and will progress to describing a number of more complex situations. GoldSim is a very powerful tool, and hence not all of GoldSim's features will be described. However, the simple examples that are shown should provide a good indication of GoldSim's range of capabilities.

In particular, we will discuss the following topics:

- Modeling Simple Failures
- Modeling Multiple Failure Modes
- Modeling the Reliability of Systems
- Modeling Repairs, Replacement and Preventive Maintenance
- Modeling Complex Interdependencies and Dynamically Changing Systems
- Modeling the Consequences of Failure (System Performance)

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 two specialized 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 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.

Modeling Simple Failures

We will begin by considering the simplest case possible: a component that has a constant failure rate (i.e., an exponential failure distribution). The reliability of such a simple component can be described using a simple closed-form equation. In fact, the reliability (i.e., the probability that the component will perform its required function over a specified time period t) can be written as follows:

$$R = e^{-\lambda t}$$

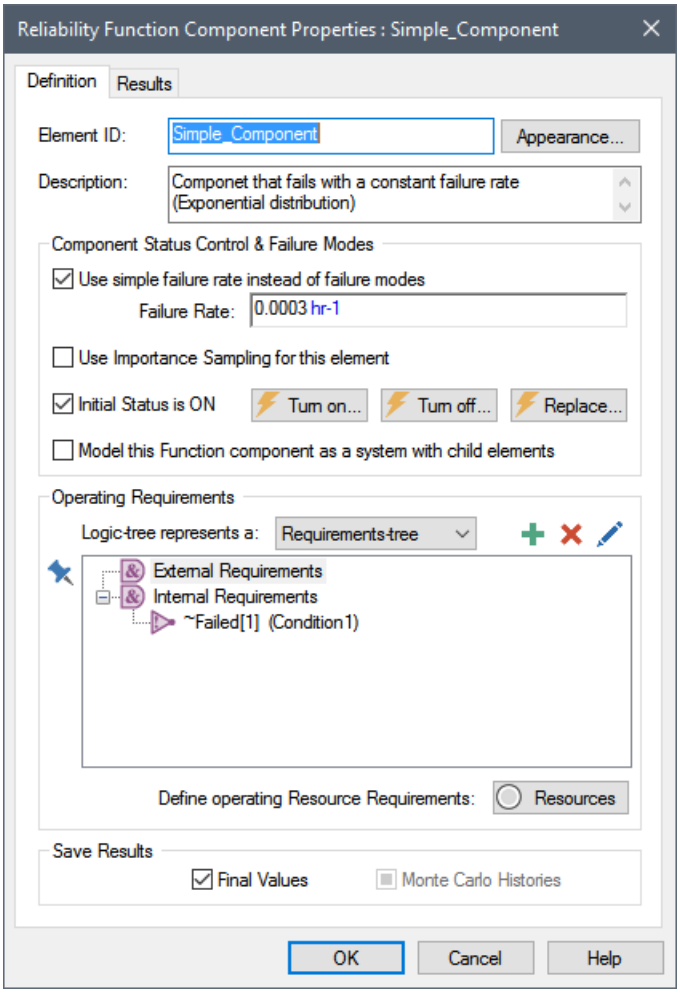
where λ is the constant failure rate. It can also be shown that the mean time to failure (MTTF) is equal to 1/λ.

So, as an example, let's consider a component with a failure rate of 0.0003 failures per operating hour. The reliability over a 300-day continuous operating period would then be:

$$R = e^{-\lambda t} = e^{-(0003\ \mathrm{hr}^{-1})(300\ \mathrm{days})(24\ \frac{\mathrm{hr}}{\mathrm{day}})} = 0.115$$

The MTTF would be 3333 hr = 139 days.

So how would we model this in GoldSim? We first specify in GoldSim how the component can fail. The component is represented by a Function element that looks like this:

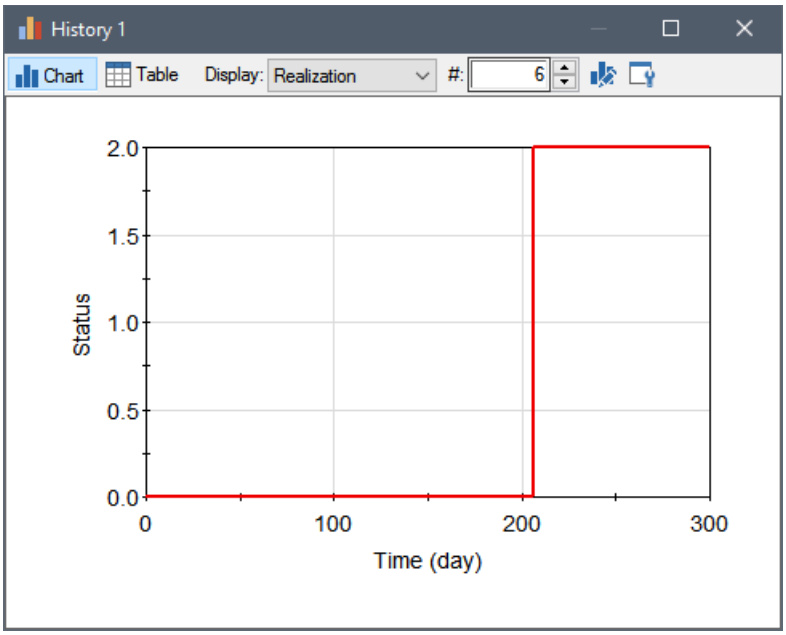


The Function element dialog has lots of options (and we will discuss some of them later), but in this case, there are only two fields that are of interest:

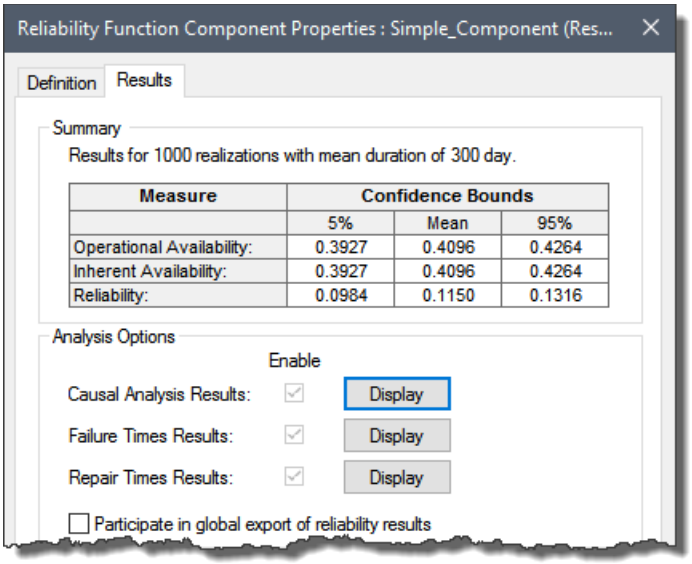
- **Use simple failure rate instead of failure modes**: This instructs GoldSim to assume a simple exponential failure distribution.
- **Failure Rate**: This is the rate of failure (i.e., the λ term in the equations above). Note that this also represents the (constant) ***hazard rate***.

Once the Function is defined, we can simulate the system. What we are going to do is run the model for the operating period of interest (300 days). During the simulation, GoldSim samples the failure distribution and determines when the component fails. Of course, the failure distribution represents the inherent randomness (i.e., the stochastic nature) of failure, so we will need to run a Monte Carlo simulation with multiple realizations to see how the component will perform statistically. In this example, we will run the model for 1000 realizations.

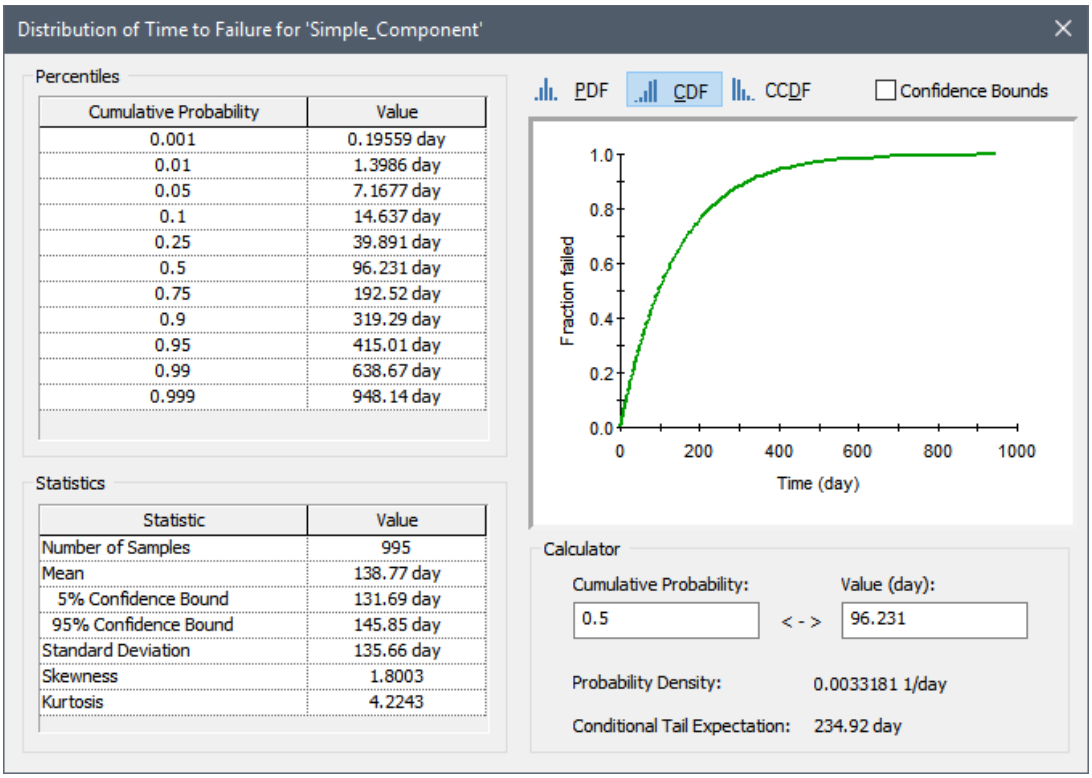
One of the basic outputs of the Function element is its ***Status*** at any given time. This is indicated by a number. For example, a status of 0 indicates that the component is operating. A status of 2 indicates that for one or more reasons, it is not. Here is a time history plot of a single realization (realization #6 of 1000) for the status for this component:



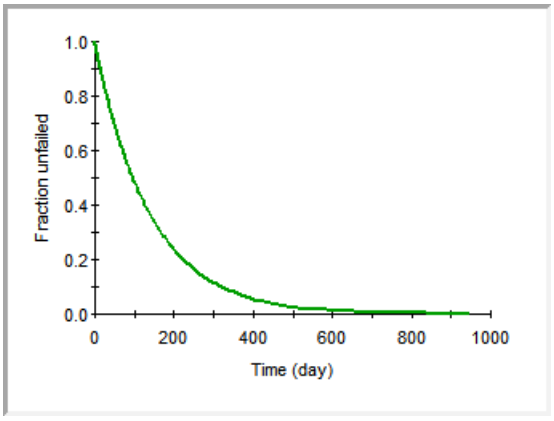
This indicates that for this realization, the component failed at just past 200 days. Given a MTTF (based on the closed-form solution) of 139 days, such a failure time is reasonable. Although as we will see later, the status of a component is very useful (e.g., we can reference it in order to realistically model complex dependencies), what we are interested in for this example are the traditional statistical reliability metrics. By collecting all of the realizations together, GoldSim automatically does so. Here are the statistical results of the simulation of this component:



Note that the **Reliability** is computed as 0.115 (consistent with the closed-form solution). The confidence bounds indicate the uncertainty in this estimate (due to the number of realizations). We'll wait to discuss the **Availability** results until we consider repairs. If we press the **Failure Times Results** button GoldSim displays the following result:



This is the simulated distribution of failures. The plot on the right is the CDF of the failure distribution. We could press the **CCDF** button to display the ***complementary cumulative distribution function*** (which is also referred to in this case as the ***reliability function***):



If you look at the **Statistics** portion of the dialog, you will note a **Mean** (i.e., the MTTF) that is consistent with the closed-form solution (i.e., 139 days).

Modeling Multiple Failure Modes

Now that we have discussed this trivial case, let's make it a bit more complicated. We will do this in two ways:

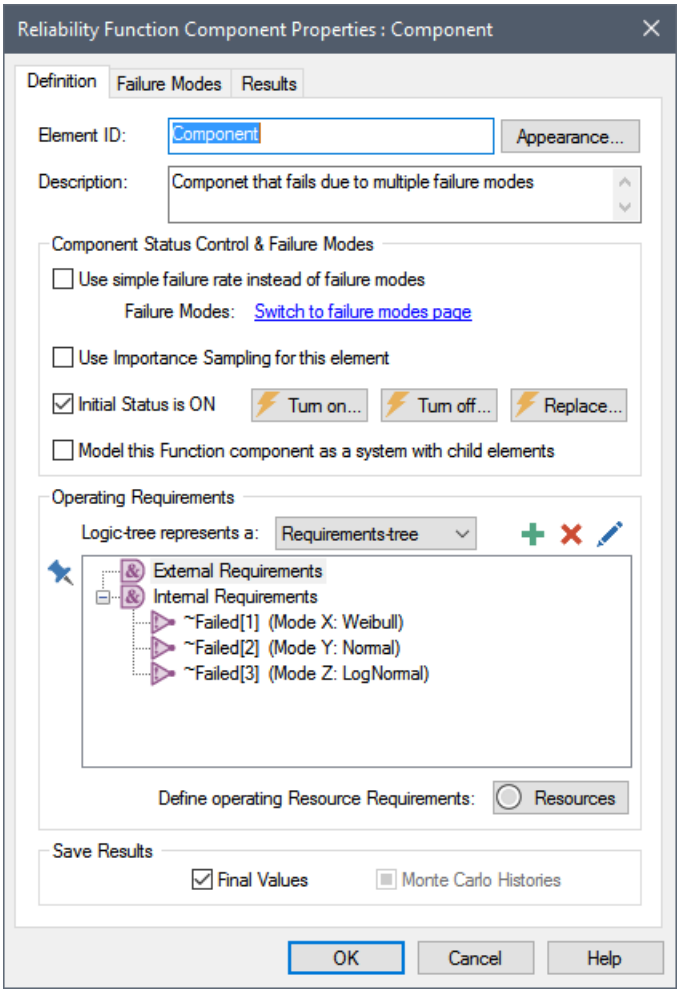
- The component can fail due to multiple modes.
- The failure modes are time-dependent. That is, unlike the exponential failure, which is memoryless, the time to failure for a particular mode is a function of how long the component has been operating (i.e., the failure rate is not constant).

If we make the further assumption that the failure modes are independent and of a particular form, such a system can still be solved using closed-form equations (the Reliability function can be computed as the product of the Reliability function for each mode). However, in this example, we will use two distributions (the Normal and LogNormal) that actually do not have closed-form solutions (although there are techniques using statistical tables to solve for these). So we won't bother to walk through the traditional calculations. Rather, we will just show how this is represented in GoldSim.

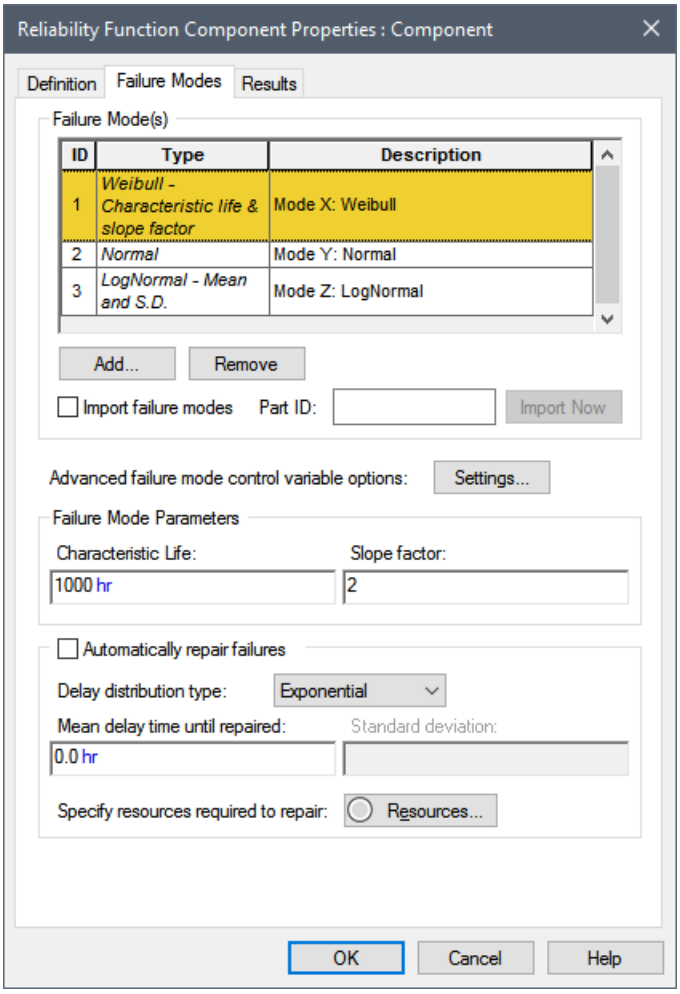
This example assumes three independent failure modes described using the following distributions:

- Weibull: Characteristic life = 1000 hrs; Shape factor (slope) = 2
- Normal: Mean life = 1200 hrs; Standard deviation = 200 hrs
- LogNormal: Mean life = 1000 hrs; Standard deviation = 100 hrs

The main page of the Function element representing the component looks like this:



Note that the checkbox labeled **Use simple failure rate instead of failure modes** is cleared. As a result, a **Failure Modes** tab is available. This is where we define the three failure modes:

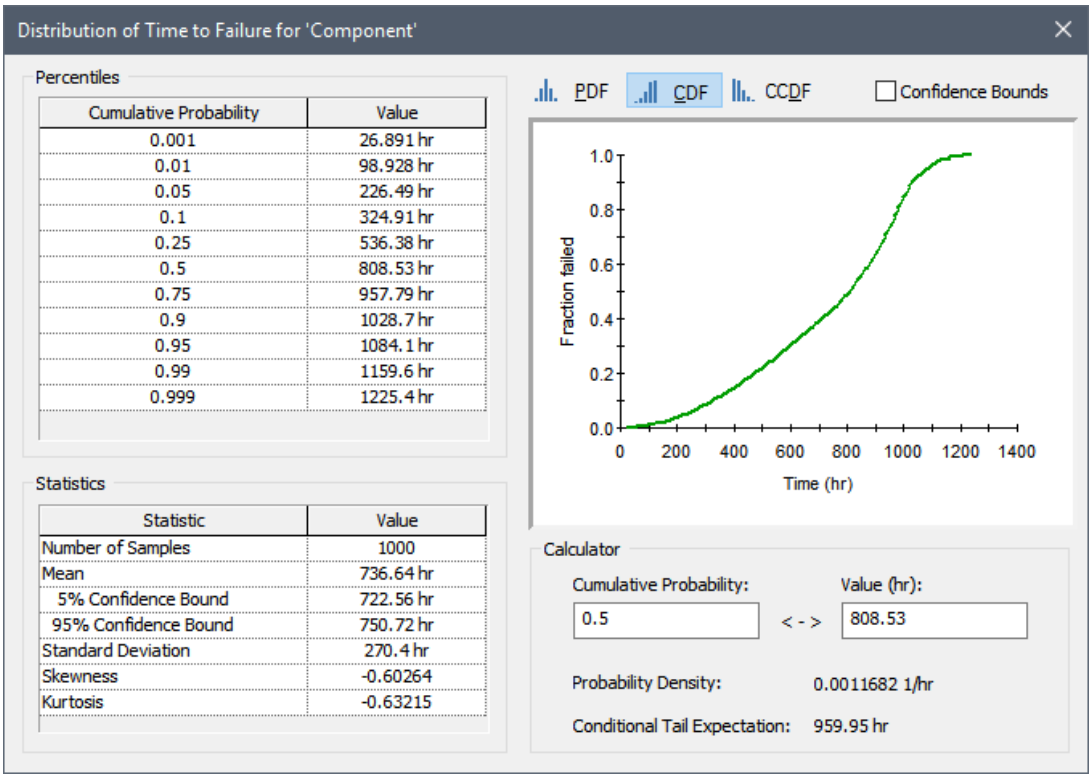


In this case, the first failure mode (the Weibull) is selected, so the **Failure Mode Parameters** section of the dialog shows the inputs for that mode. The inputs for the other two modes could be accessed by selecting them at the top of the dialog.

Note: In a real model, we would not enter the parameters directly here as numbers. Rather, we would define other elements and reference the element names here. If we were uncertain about the parameters, we could define them as probability distributions to represent this uncertainty.

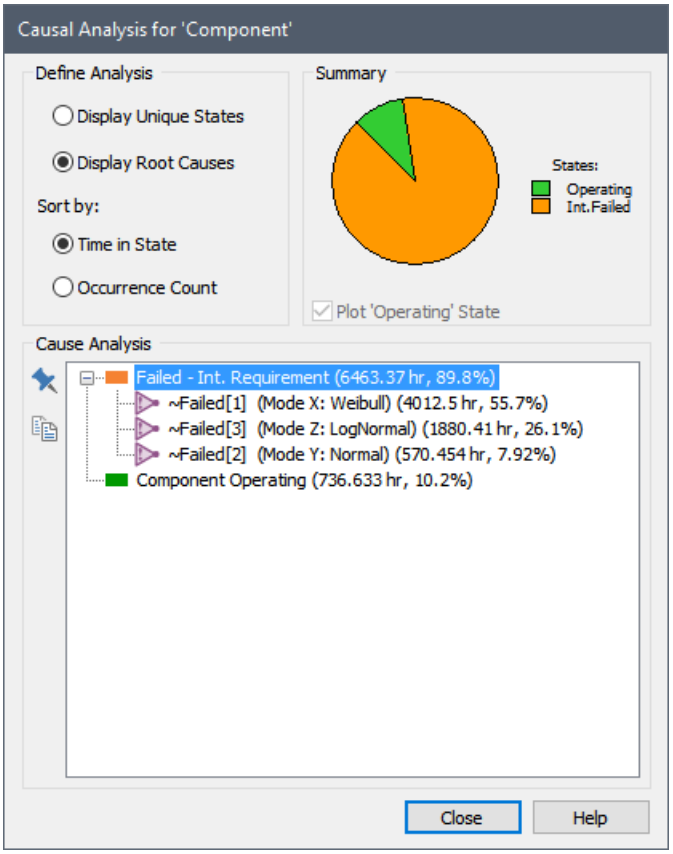
Note: By specifying multiple failure modes (e.g., early failures, random failures and wear out failures) and taking advantage of some advanced dynamic failure mode features, you can readily represent a "bathtub" failure curve.

If we run this model (again, for 300 days and 1000 realizations) and look at the results, the combined failure distribution looks like this:



Note the rather complex shape. Note also that the MTTF is less than the mean of any of the individual modes.

More interestingly, we can view a ***root cause analysis*** for the component to see which modes cause failures:



This indicates that 56% of the failures were due to the Weibull failure mode, 26% were due to the LogNormal failure mode, and 8% were due to the Normal failure mode.

Modeling the Reliability of Systems

In the examples we have just discussed, we were considering a single component. Of course, in the real world, systems consist of multiple components (e.g., a computer system consists of multiple components, such as a hard drive, a power supply and a CPU). Depending on the configuration of the components in the system (i.e., the system dependencies), a failure in one component may or may not result in the failure of the system. The various configurations can be illustrated in the form of ***reliability block diagrams***. In traditional approaches, depending on the complexity of the configuration and assumptions (e.g., independence), it may be possible to solve for these using closed-form solutions. We'll consider several reliability block diagrams below, and illustrate how GoldSim can very readily represent any configuration.

Serial Systems

Let's first consider the following simple system consisting of three components in series:

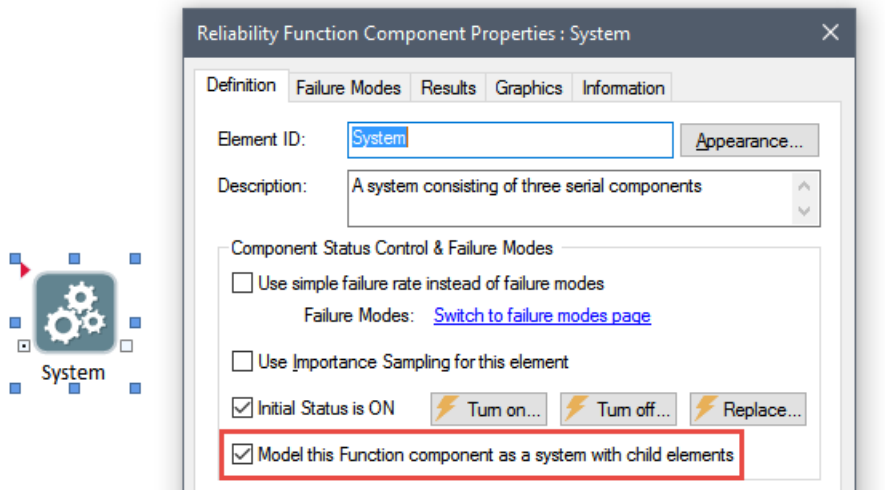
In this configuration, Component B requires Component A to be operating and Component C requires Component B to be operating. Hence, all components must function for the system to function.

Let's further assume that each component fails according to a single mode:

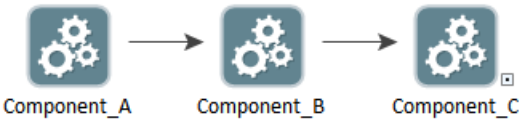
- **Component A**: Weibull distribution with Characteristic life = 1000 hrs and Shape factor = 2
- **Component B**: Normal distribution with Mean life = 1200 hrs and Standard deviation = 200 hrs
- **Component C**: LogNormal distribution with Mean life = 1000 hrs and Standard deviation = 100 hrs

To represent systems in GoldSim, we are going to take advantage of a capability we discussed earlier in this paper: the ability to create sub-systems using Containers. This capability makes it very easy for GoldSim to represent any kind of system configuration in an intuitive manner.

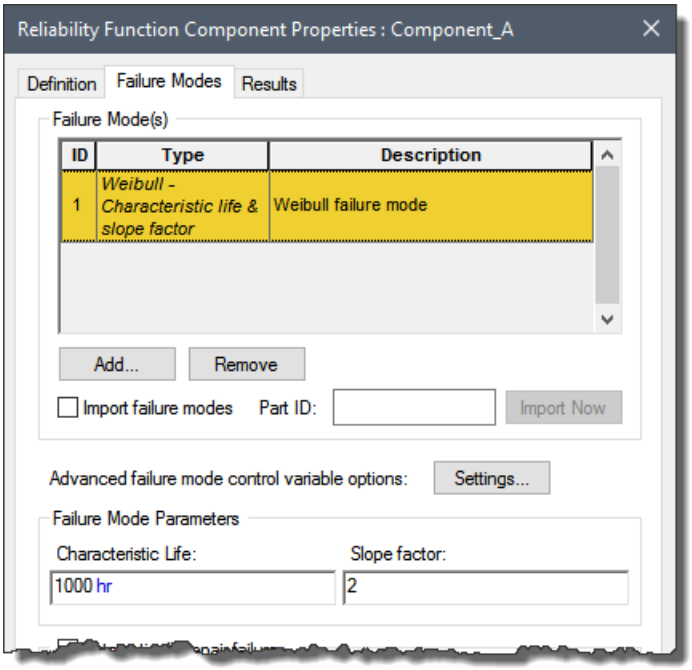
The GoldSim model for this consists of four elements: a Function element representing the entire system, and a Function element for each component. We first create a Function element and instruct GoldSim to treat it as a system (i.e., a Container):



We can then "enter" this element (by clicking the small red triangle). Inside this element we then see the three components:

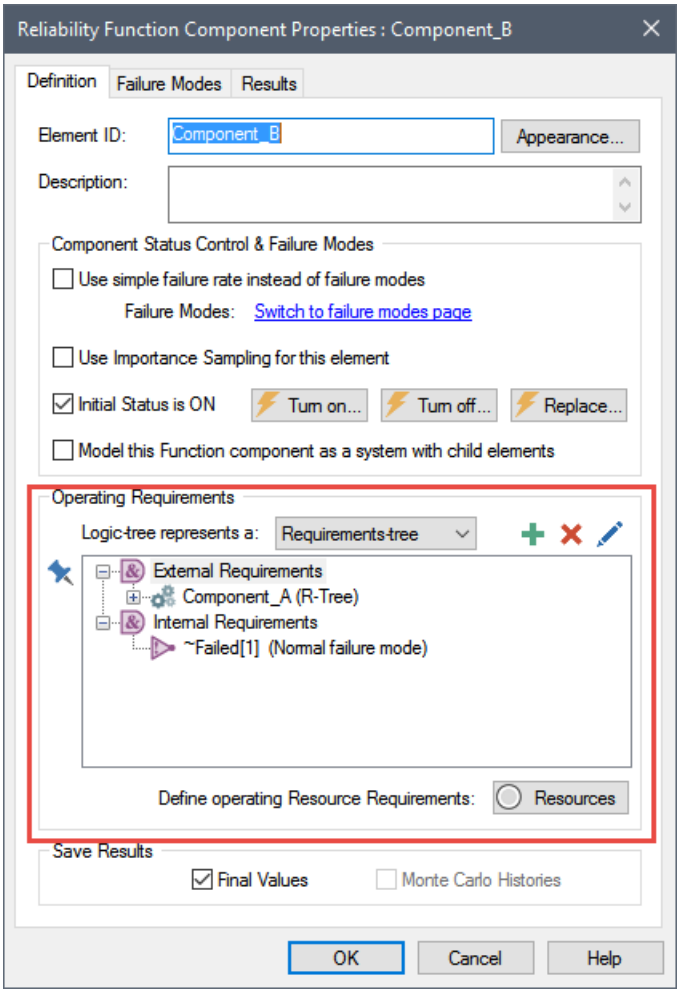


If we were to examine any of these, we would see a single failure mode defined for each. Here, for example, is Component A:



Note that the three components are visually connected (via influences). This is because their dependencies (i.e., Component B requires Component A to be operating and Component C requires Component B to be operating) have been specified by defining ***Operating Requirements*** for Component B and Component C.

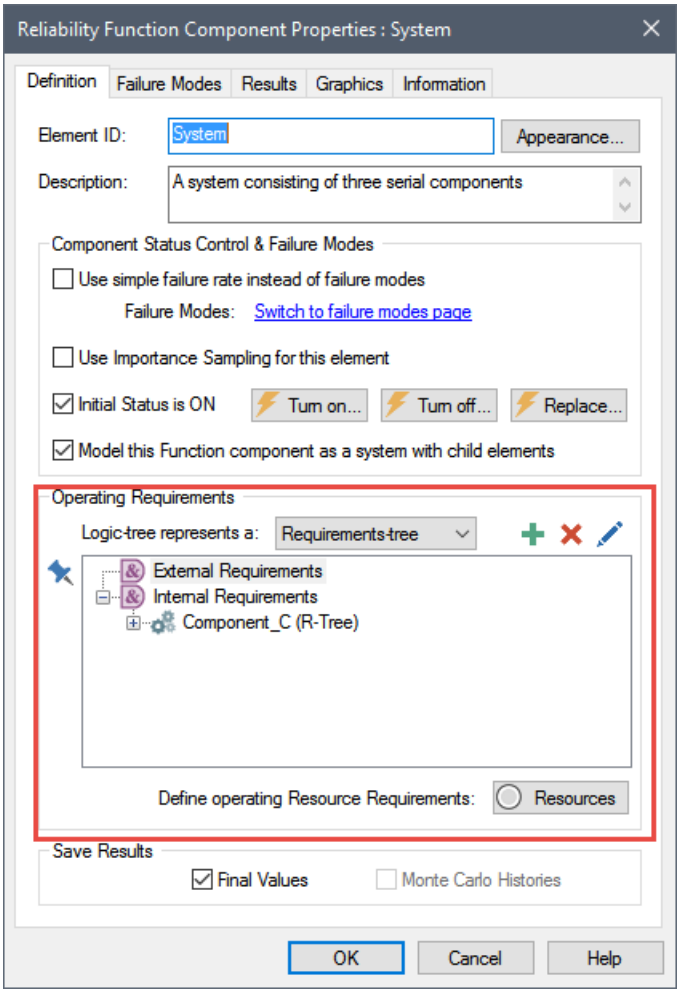
To see this, let's look at Component B:



GoldSim allows you to create a ***Requirements tree*** (or optionally, the opposite, a ***Fault tree***) to define the Operating Requirements for the element. There are two types of Operating Requirements: ***External Requirements*** and ***Internal Requirements***. In order for the component to operate, External and Internal Requirements must both be met. External Requirements are "outside" of the component itself. In this case, the sole External Requirement for Component B is that Component A must be operating. Internal Requirements are "inside" the component. In this case, the sole Internal Requirement for Component B is that the component itself is not failed (due to its specified failure mode).

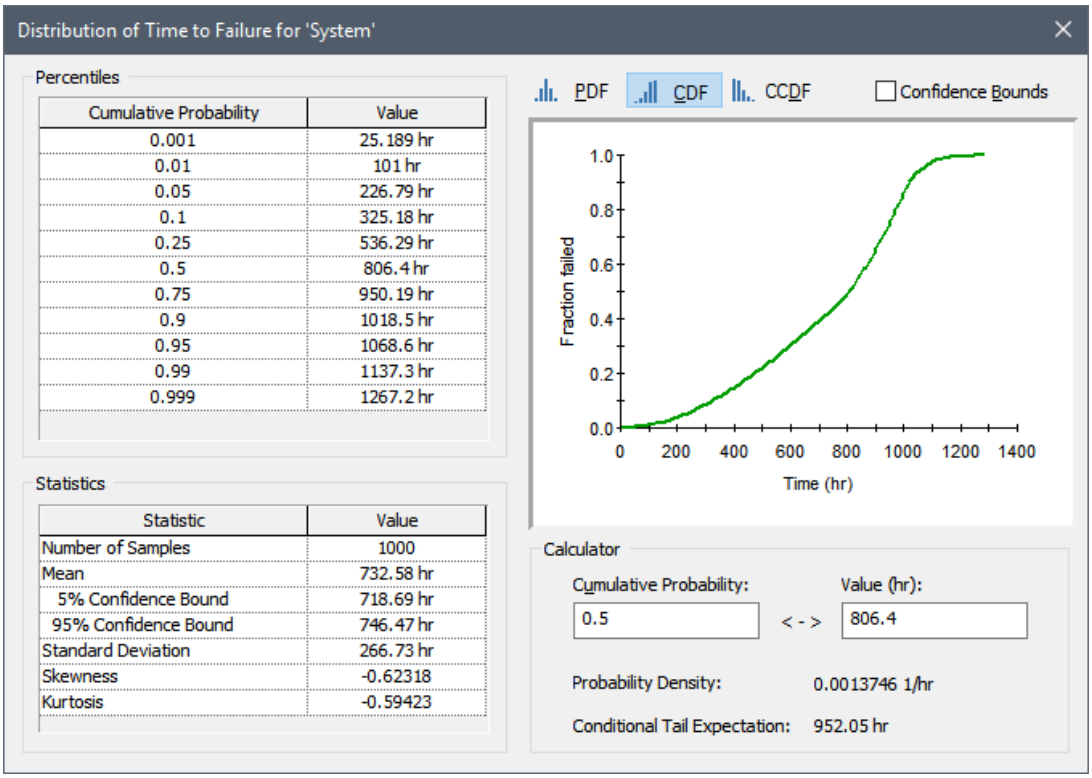
Component C is defined similarly, with an External Requirement that Component B must be operating and an Internal Requirement that the component itself is not failed (due to its specified failure mode).

If we were to run the model, we could look at failure distributions and reliability metrics for each of the three components. But that is not what we are interested in. We want to look at the combined failure distribution and reliability metrics for the entire system. We can do this by examining the System element itself (and looking at its results). If we look at the System element, we will note that it has no External Requirements, and no failure modes of its own:



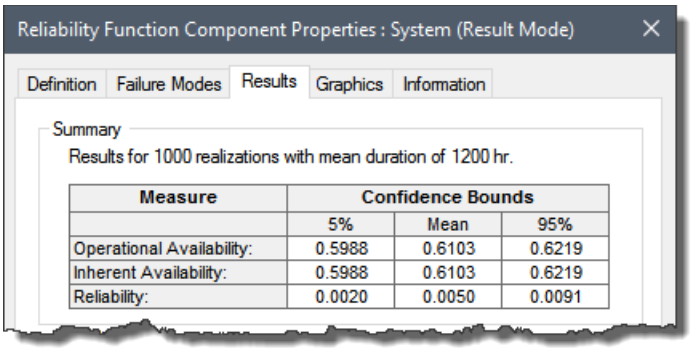
Instead, it simply has a single Internal Requirement: that Component C be operating. If Component C is operating, the system is operating. If Component C is not operating (because it has failed, or because Component A or Component B have failed), the system has failed. Note that this is considered to be an "Internal Requirement" because Component C actually exists inside of the System element.

If we run this model (for 50 days and 1000 realizations) and look at the results for the System, the failure distribution looks like this:



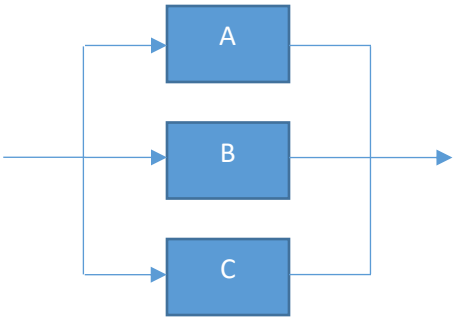
This should look familiar. It is statistically identical to the model with a single component and three failure modes. That is, a system consisting of three independent serial components is mathematically identical to a component with three independent failure modes.

The **Reliability** for this system is close to zero (the probability of system failing over 50 days is almost 100%):



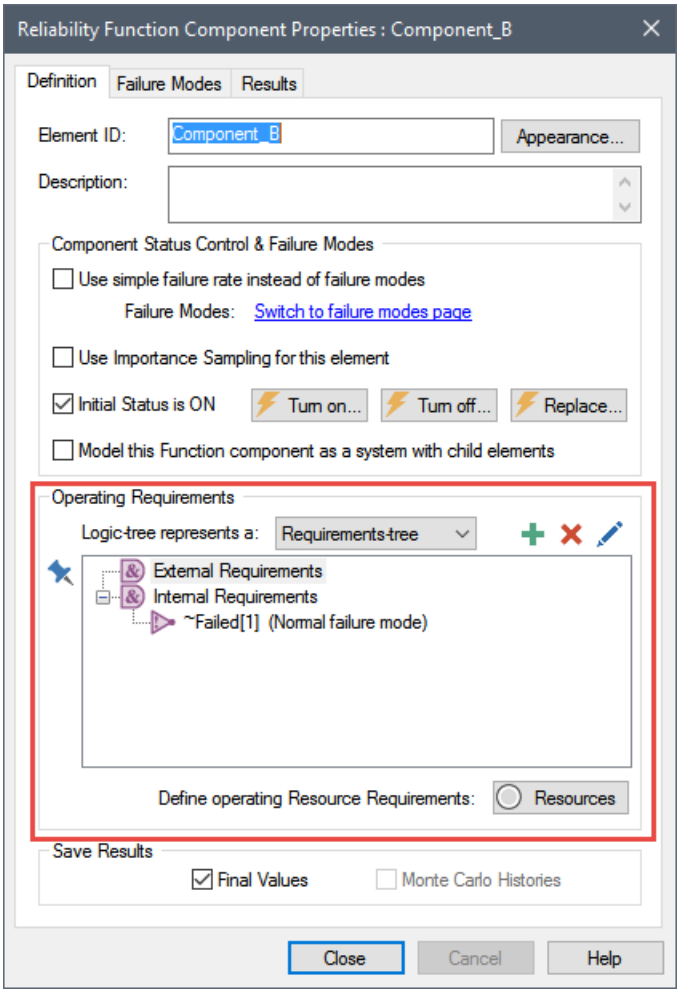
Parallel Systems

Let's now consider the same system, but assume that the three components are in parallel:



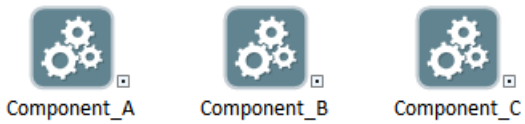
In this configuration, only one of the components must function for the system to function (i.e., they are redundant).

To modify the previous model to represent this configuration, we need to make two changes to the model. First, we must remove the External Requirements that link Component A to Component B and Component B to Component C. So, for example, Component B now looks like this:

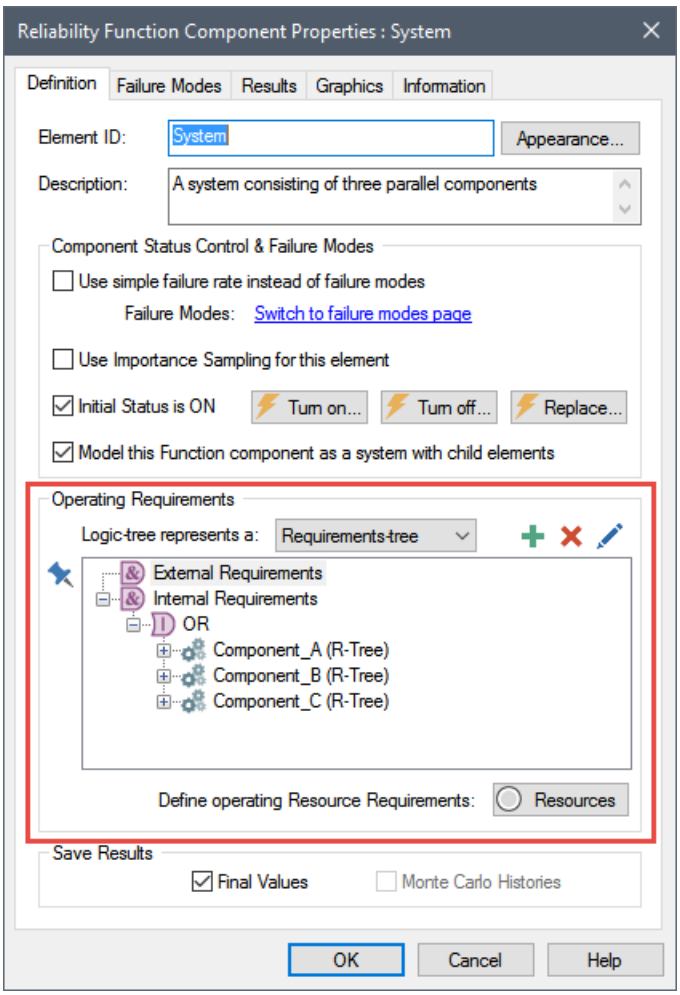


The only **Operating Requirement** for Component B is an Internal Requirement that the component itself is not failed (due to its specified failure mode).

As a result of these changes, the three components are no longer linked together by influences (since they have no dependencies on each other):

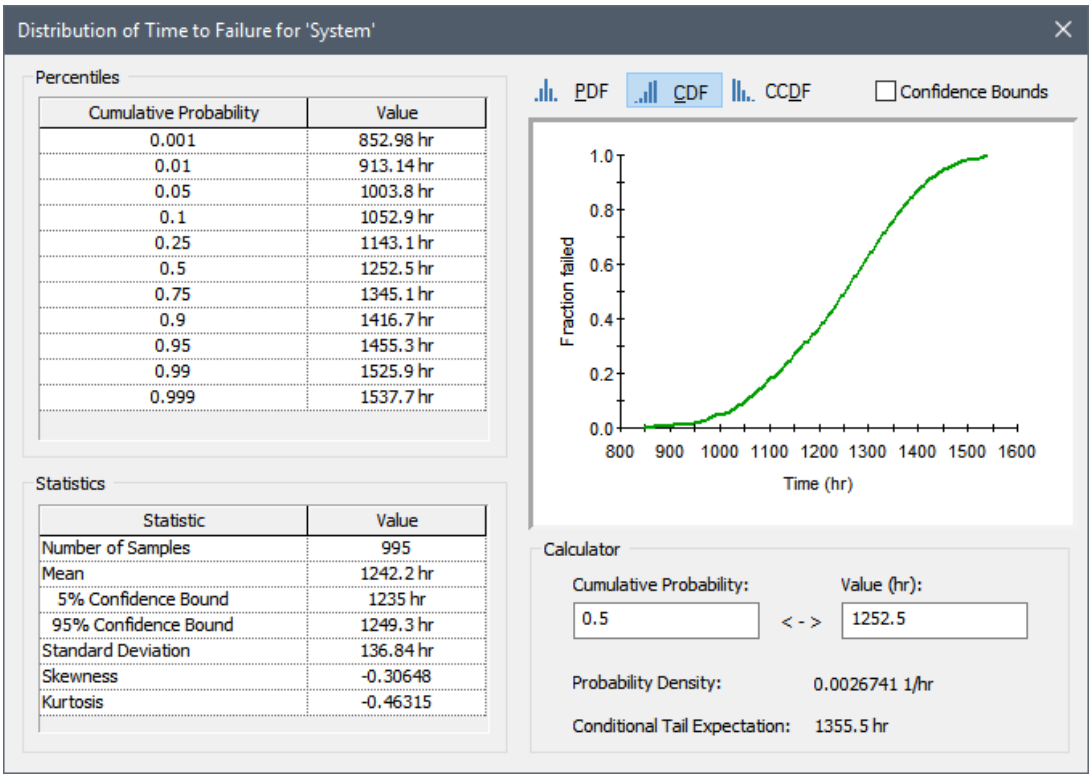


To represent that fact that the system itself requires one of the three components to be operating, we simply change the **Operating Requirements** for the System element. Instead of specifying that the System is operating if C is operating, we specify that the System is operating if any of the components is operating. This is done by using an OR gate in the Requirements tree:

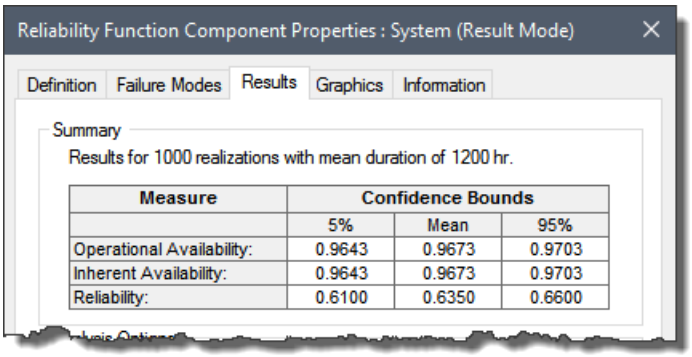


In this case, we have specified **Operating Requirements** consisting of a Requirements tree that specifies that in order for the System to operate, Component_A, Component_B *or* Component_C must be operating (they are all listed under an *OR* gate in the tree).

If we run this model (again, for 50 days and 1000 realizations) and look at the results for the System, the failure distribution looks like this:



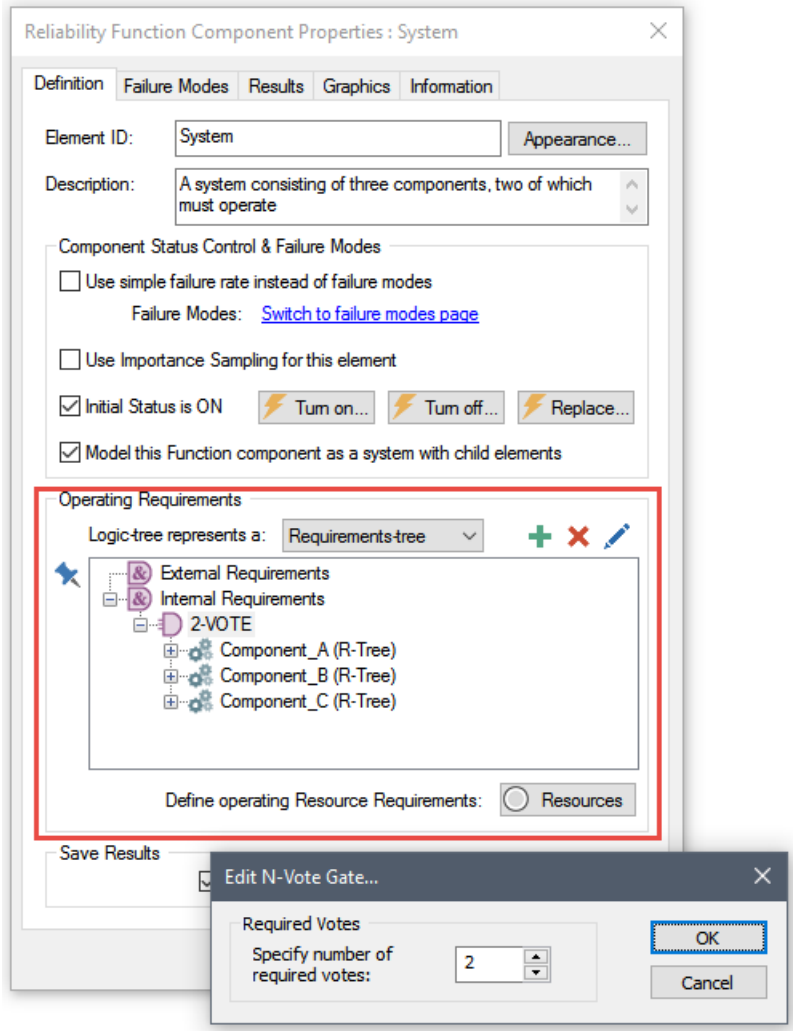
As we would expect, the MTTF for the system is significantly higher (and the shape of the distribution is very different). Moreover, due to the redundant nature of the system, the Reliability is now about 64% (instead of essentially zero):



k-out-of-n Redundancy

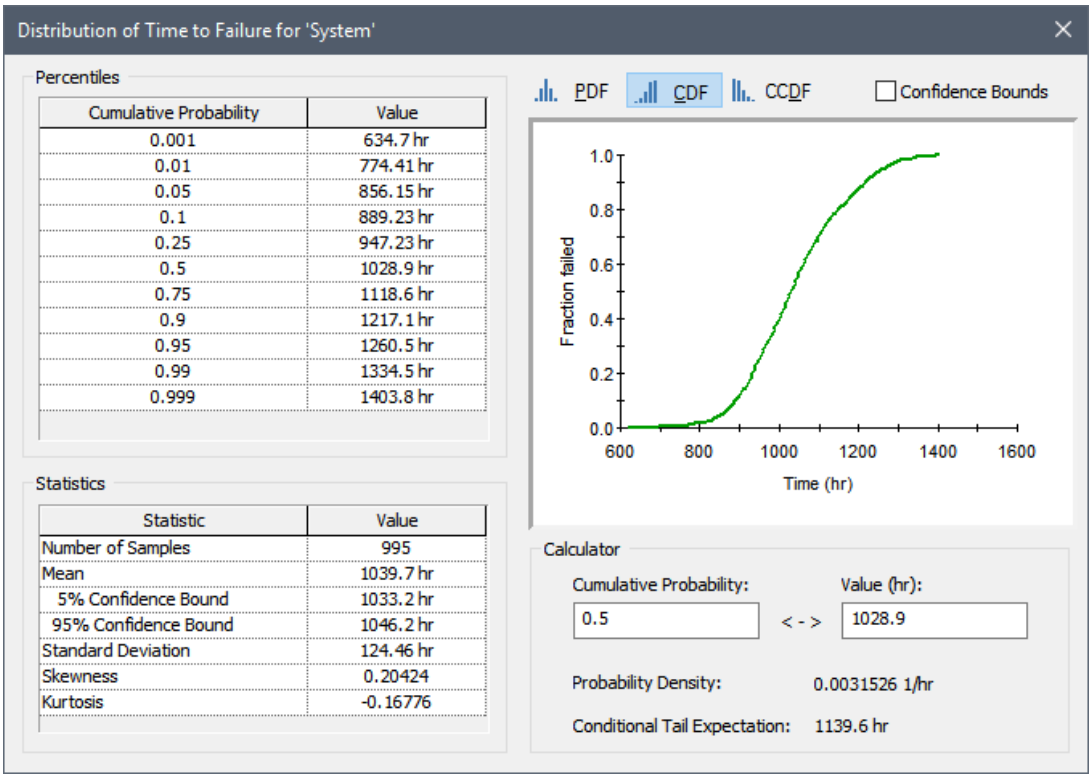
Let's now consider the same system, but assume that there is a "2-out-of-3" redundancy among the three components. In particular, we assume that two of three components must be operating in order for the system to operate. We would expect the performance of this system to be somewhere between the serial system (all must be operating) and the parallel system (one must be operating).

To modify the previous model to represent this configuration, we simply need to change the OR gate in the Requirements tree for the System element to an ***N-VOTE*** gate:

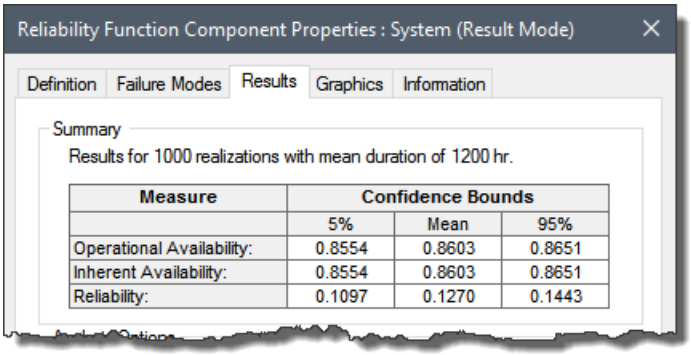


In this case, we have specified **Operating Requirements** consisting of a Requirements tree that specifies that in order for the System to operate any two of the components must be operating (they are all listed under a 2-VOTE gate in the tree).

If we run this model (again, for 50 days and 1000 realizations) and look at the results for the System, the failure distribution looks like this:



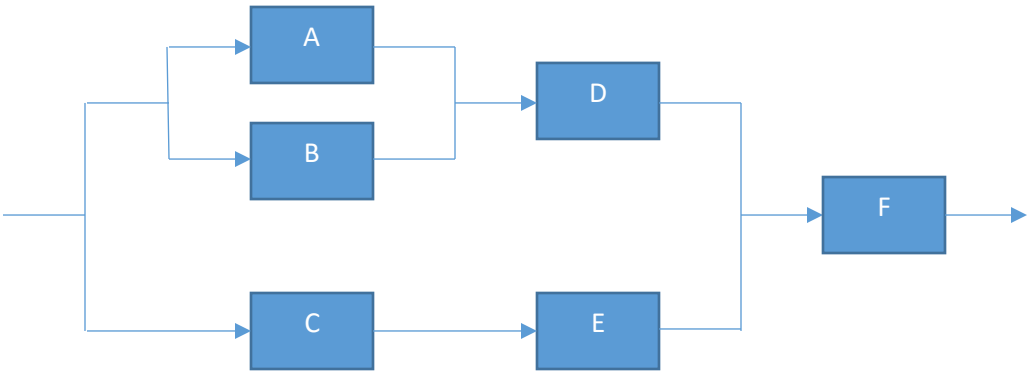
As we would expect, the MTTF for the system is between that for the parallel and series systems. This is also the case for the Reliability:



Combined Series-Parallel Systems and Other Complex Configurations

The discussions above considered very simple configurations. Let's briefly consider some more complex configurations in order to see how they would be represented in GoldSim.

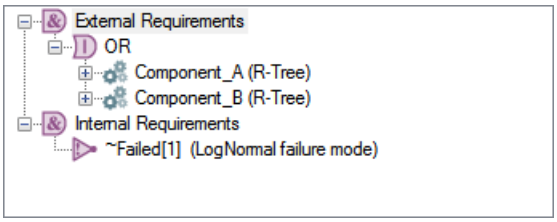
First, let's consider the following system:



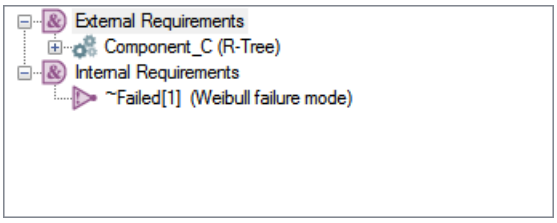
How would we represent this system in GoldSim? The approach is straightforward and very easy to implement. First we would create the six components (Function elements) and place them inside another Function element that was specified to be a System (i.e., a Container). Those six components would have one or more failure modes. The dependencies illustrated in the diagram are then specified by defining appropriate External Requirements for the six elements.

Components A, B and C would have no External Requirements.

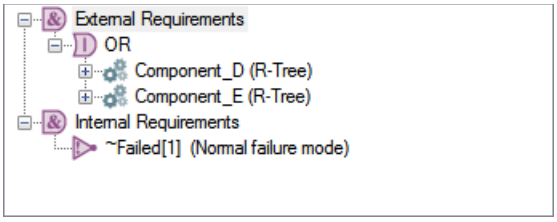
The External Requirements for Component D would be an OR gate:



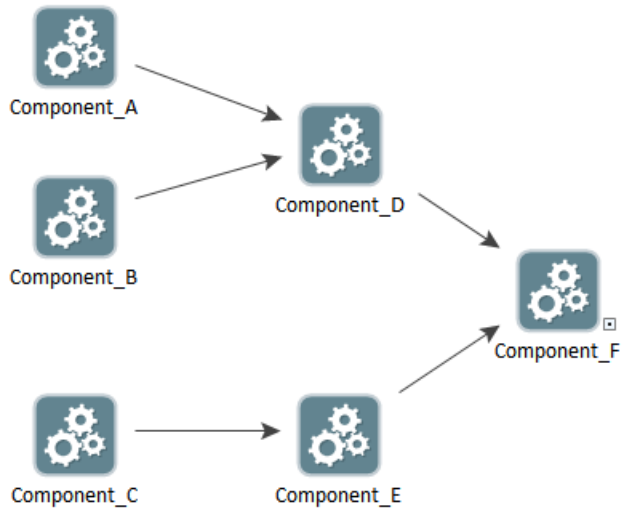
The External Requirements for Component E would be a simple dependence on Component C:



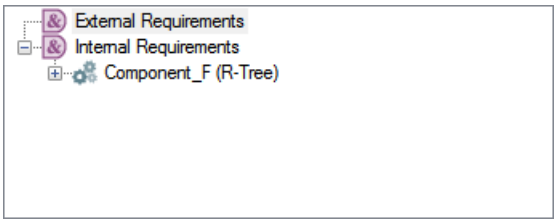
The External Requirements for Component F would be an OR gate:



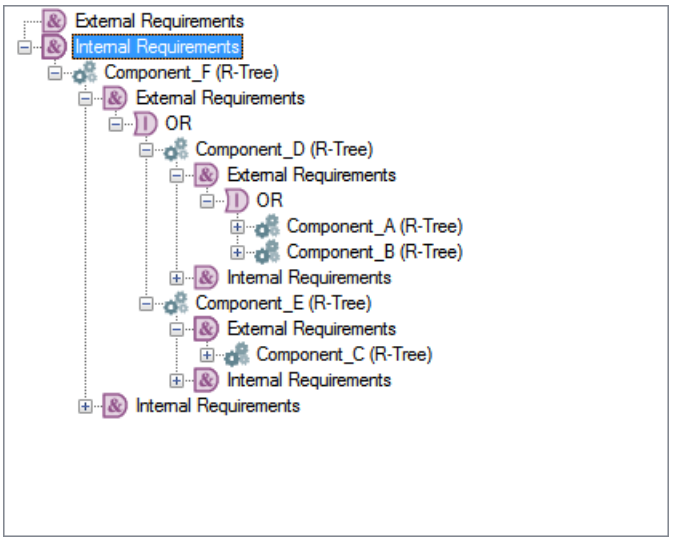
This would result in the following influences to be drawn between the components:



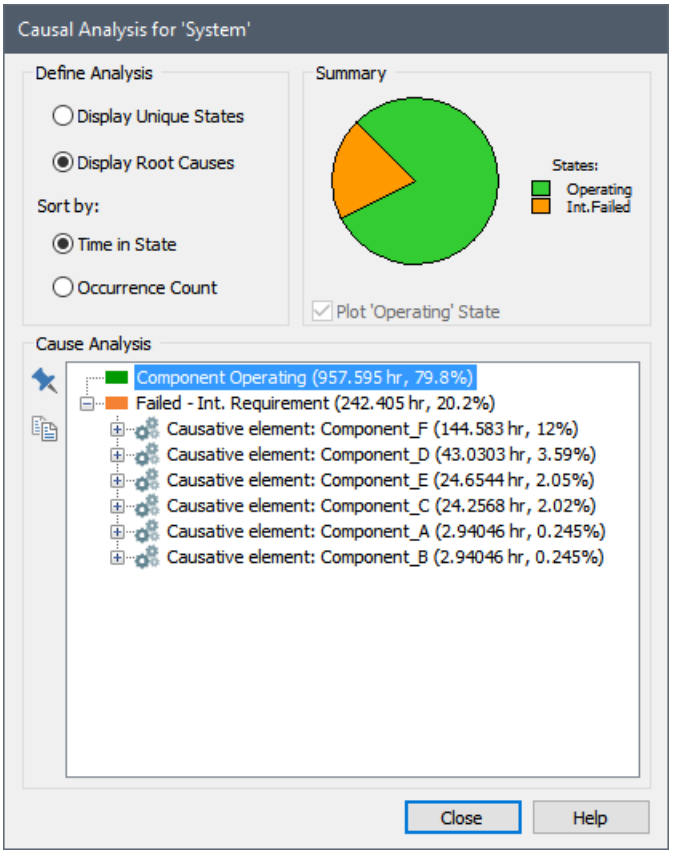
Finally, the System itself would have a single Internal Requirement: that Component F be operating:



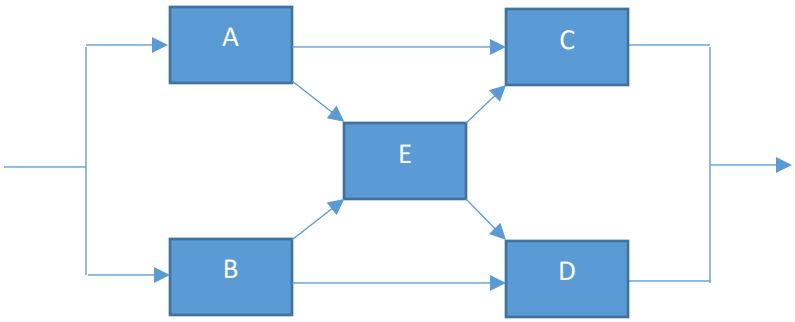
Note that the Requirements tree for the Function element representing the System (shown above) can be expanded to see the full requirements tree for all components inside the System:



If we run this model, we can view a causal analysis to see which components cause failure of the System:



Let's now consider one additional configuration, one that cannot be decomposed into series and parallel relationships:

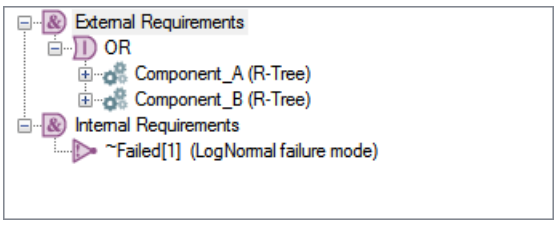


Based on the previous discussions, it should be obvious how this would be represented in GoldSim.

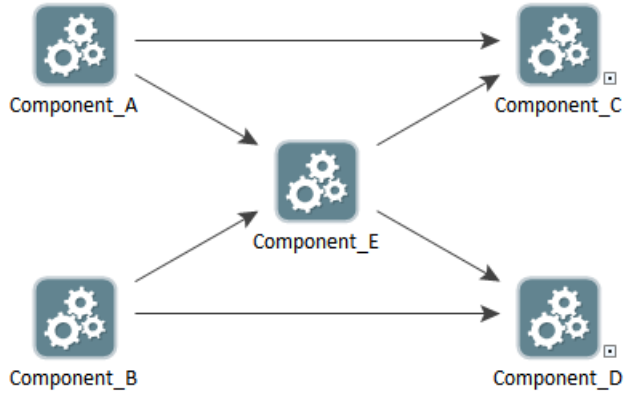
First we would create the five components (Function elements) and place them inside another Function element that was specified to be a System (i.e., a Container). Those five components would have one or more failure modes. The dependencies are then specified by defining appropriate External Requirements for the five elements.

Components A and B would have no External Requirements.

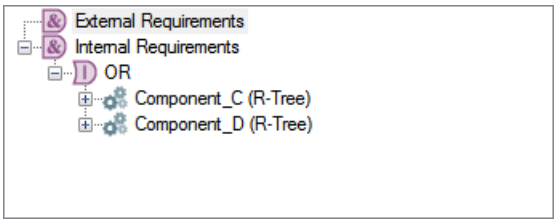
The External Requirements for the other three components would all be OR gates. Here, for example, are the External Requirements for Component E:



This would result in the following influences to be drawn between the components:



Finally, the System itself would have an Internal Requirement that was an OR gate:

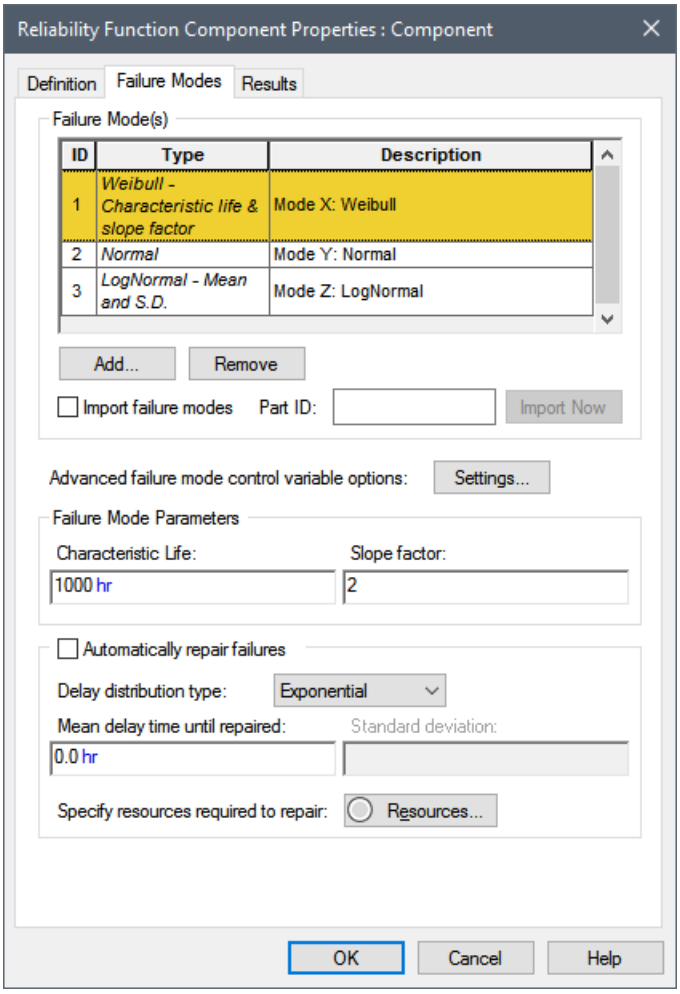


Modeling Repairs, Replacement and Preventive Maintenance

In the previous sections we discussed how failure could be modeled in GoldSim. In many systems failures can be repaired (or components completely replaced), and in order to model the actual performance of the system (e.g., the availability), we need to be able to represent these repairs. Moreover, because designing an effective preventive maintenance program is one of the more powerful applications of reliability modeling, we want to be able to model such a program. Below we show how GoldSim can readily represent repairs, replacement and preventive maintenance.

Modeling the Repair of Failure Modes

Recall the model that we discussed earlier that involved a single component with three failure modes. We noted that the component had a **Failure Modes** tab that looked like this:

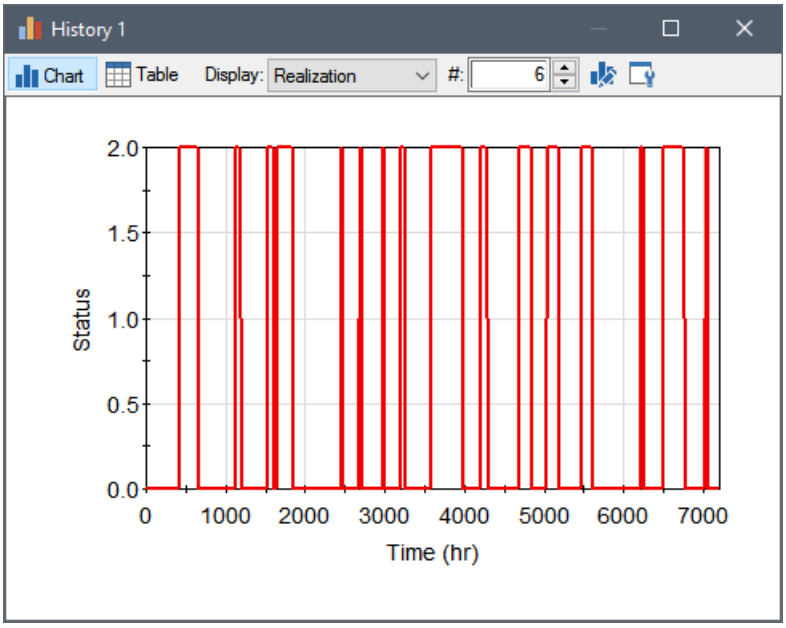


In this case, the first of the three failure modes (the Weibull) is selected, so the **Failure Mode Parameters** section of the dialog shows the inputs for that mode. The inputs for the other two modes could be accessed by selecting them at the top of the dialog.

Note at the bottom of the dialog there is an option to **Automatically repair failures**. If we check this box, we can define a repair time distribution for the failure mode (as either an Exponential, Gamma or LogNormal). If a failure mode is set to automatically repair failures, when a failure occurs due to that mode, the repair time is sampled from the distribution, and after the time passes, the failure is considered to be repaired (and if the component has not simultaneously failed due to other modes, it becomes operable again). Each failure mode can be assigned a different repair time distribution.

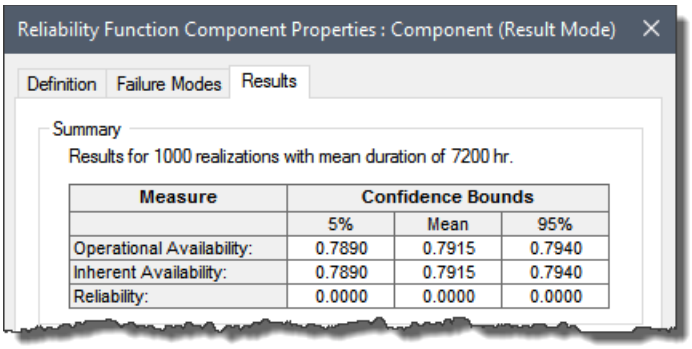
In this model, we will assign Exponential repair time distributions for each of the three failure modes, with mean repair times of 100 hr, 150 hr, and 50 hr, respectively.

After running the model (for 300 days and 1000 realizations), we can plot the Status of the component. Recall that the Status is represented by an integer, and a Status of 0 indicates that the component is operating and a Status of 2 indicates that for one or more reasons, it is not. Here is a time history plot of a single realization (realization #6 of 1000) for the Status for this component:



As can be seen, the component repeatedly fails and is repaired throughout the simulation. Due to the differences in the repair time distributions for each of the failure modes (and the fact that the times are sampled from distributions), we see a variability in the time to repair a failure.

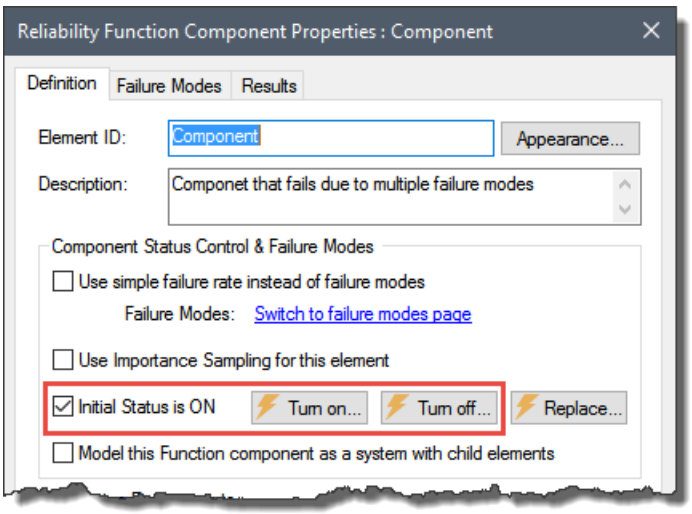
Let's now look at the statistical results for all 1000 realizations:



Measure	Confidence Bounds		
	5%	Mean	95%
Operational Availability:	0.7890	0.7915	0.7940
Inherent Availability:	0.7890	0.7915	0.7940
Reliability:	0.0000	0.0000	0.0000

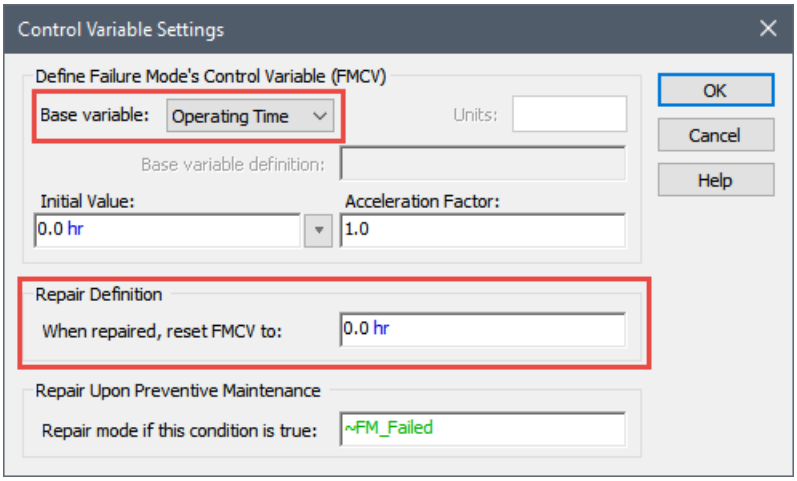
The **Reliability** of this component it zero (it never survives for 300 days), but the **Availability** is about 79%. That is, it is operating about 79% of the time.

Note that GoldSim computes two different Availabilities. The **Operational Availability** represents the fraction of time the component has been operating over the simulated time. The **Inherent Availability** represents the fraction of time the component has been operable over the simulated time. In this simple model, these are the same. However, in many models they will be different. This is because a component could be operable (unfailed), but may not be operating. There are a number of reasons that this could be the case. For example, we could choose to define events that turn a component off and on (e.g., perhaps it only operates during certain shifts):



If we did so, the Operational Availability would be smaller than the Inherent Availability because the component could be operable (unfailed), but would not be operating for certain periods because it was turned off.

When repairing a failure mode, GoldSim provides you with the ability to define exactly what a repair means. Each failure mode has a dialog to define the ***Failure Mode Control Variable*** (FMCV):



The FMCV is the variable that is referenced by the failure mode to determine when failure occurs (i.e., the control variable represents the x-axis of a failure distribution plot). It defaults to Operating Time, but can also be specified as Total Time or, as we will see later, to a user-defined metric such as mileage.

In this dialog, you can specify what happens during a repair (i.e., what the FMCV is reset to upon repair). Resetting the FMCV to zero is equivalent to replacement (i.e., making it as good as new). But you could also set it to a positive value (e.g., using a refurbished part that already has some wear on it).

Modeling Replacement and Preventive Maintenance

GoldSim provides the ability to model maintenance in two different ways:

- You can schedule a periodic replacement of the entire component (which repairs all failures and resets the FMCV for all failure modes to zero).

- You can schedule a periodic preventive maintenance. When you do so, you can specify that the maintenance only impacts certain failure modes and/or only resets their FMCVs to specified values.

By doing so, you can run simulations to predict how different maintenance regimes impact system performance.

Modeling Complex Interdependencies and Dynamically Changing Systems

In the sections above, we illustrated how GoldSim can readily model the failure and repair of systems of components. The great power of GoldSim's simulation-based approach, however, is its ability to represent systems that cannot easily be represented by traditional approaches, including:

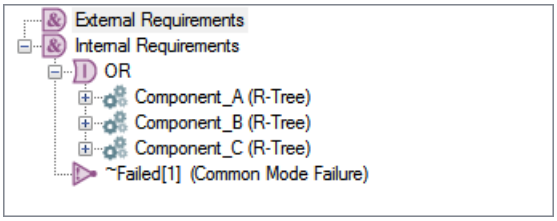
- Systems that can be impacted by external environmental processes, and whose properties can change suddenly or gradually as the simulation progresses; and
- Systems that have complex interdependencies, such as a situation where the failure of one component causes another component to wear more quickly or non-fatal failures (i.e., failure modes that only partially degrade the performance of a component).

This section will briefly illustrate the power and flexibility that GoldSim provides in this regard by discussing a number of examples of such systems.

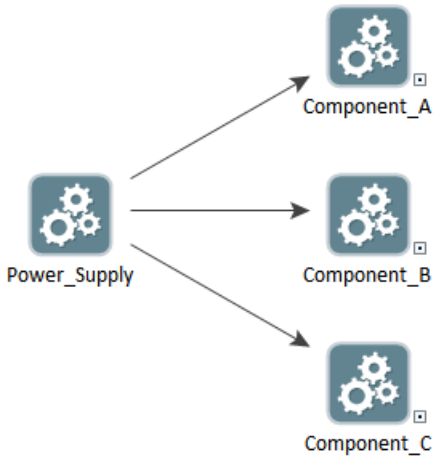
Common Mode Failures

Common mode failures are used to represent the fact that the failure rates of different components may not be independent. There are a number of factors that could cause such a dependence, ranging from the components sharing the same power supply to components responding to external environmental conditions in the same manner.

When this is treated in traditional methods it is often treated in a very simplistic way (e.g., by adding a "common mode" failure in series with those components sharing that failure mode). It is straightforward for GoldSim to handle the system in such a simple way. For example, if you had three parallel (redundant) components, you could simply include them inside a System (as we did previously), and then assign the common-mode failure to the entire System:



If the components all depended on a common component (e.g., a power supply), this would also be straightforward. The way to represent this in GoldSim would be to simply create a dependency between a power supply component and each of those components (such that if the power supply failed, the components all simultaneously failed):



Both of these methods cause the system to fail in response to a failure that affects all three components simultaneously.

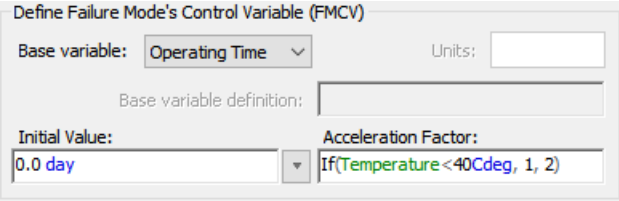
A more realistic representation of a dependence between failure modes (that is impossible to address at all using traditional methods) is that failure rates for multiple components may be simultaneously accelerated (e.g., due to environmental conditions), but the components do not necessarily fail at the same time as a result. We discuss that below.

Responding to Evolving Operational Environments

In many cases, a failure mode may be affected by dynamically changing environmental factors. For example, the wear on a component might be accelerated in hot environments. Moreover, it is possible for multiple components to be impacted by the same factor.

GoldSim provides a powerful way to represent this. For each failure mode, you can specify an ***Acceleration Factor***. The Acceleration Factor is a non-negative real number which multiplies the actual change in the base variable (e.g., Operating Time) to arrive at the failure mode's current "age" (i.e., it changes the failure rate). 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).

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 simply specify the following expression in the **Acceleration Factor** field:



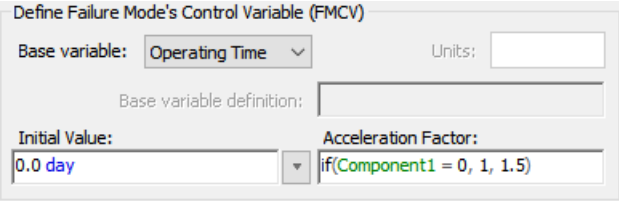
If two components had a similar dependency, they would not necessarily fail at the same time, but both of their failure rates would be accelerated at high temperature.

Load Sharing Systems

Another example of dynamic failure behavior is that associated with a ***load sharing system***. A simple example of such a system is one in which two components act in parallel (i.e., are redundant), but if one component fails, the failure rate of the other component increases as a result of the additional load placed on it.

Traditionally, this would be handled using a Markov analysis (which would be straightforward in this simple case, but would get much more difficult if additional components and states needed to be considered).

In GoldSim, this can be represented very simply using the same **Acceleration Factor** discussed in the previous section. For example, let's assume that if one of the components failed, we want the failure rate for the other component to increase by 50%. We could represent this by defining an **Acceleration Factor** for the failure modes for Component2 as follows:



Note that it references the ***Status*** of the other component (Component1). Recall that the main output of a Function element is its Status. The Status takes on an integer value throughout the simulation (e.g., 0 if operating, 2 if failed; 4 if turned off, etc.). In this case, we are saying that if Component1 is operating, there is no acceleration; if it is not operating, the **Acceleration Factor** is 1.5 (failure is accelerated by 50%).

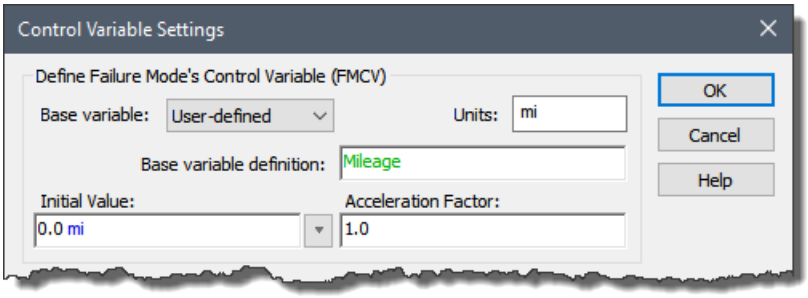
Of course, Component1 would have a similar reference for its **Acceleration Factor** (it would reference the status of Component2). Hence, representing such a complex dependency in GoldSim is easy and intuitive.

Using Physically-Based Failure Mode Control Variables

As pointed out previously, in GoldSim each failure mode for a component has a dialog to define the ***Failure Mode Control Variable*** (FMCV). The FMCV is the variable that is referenced by the failure mode to determine when failure occurs (i.e., the control variable represents the x-axis of a failure distribution plot). It defaults to Operating Time. It can also be specified as Total Time (which differs from Operating Time due to failures, as well as components that can be turned off). For certain types of components, the FMCV can also represent the number of cycles (number of landings, number of times turned on, etc.).

In addition, you can define a custom, user-defined FMCV. This is important because in some cases, it may be appropriate to define a failure control variable that is defined with respect to a physically-based variable such as mileage or perhaps the cumulative load. Any monotonically increasing function can be specified as a base variable. Because GoldSim is a flexible and powerful dynamic simulator, it can easily model and track such variables (recall the beginning of this paper when we illustrated how GoldSim could track the amount of water in a pond).

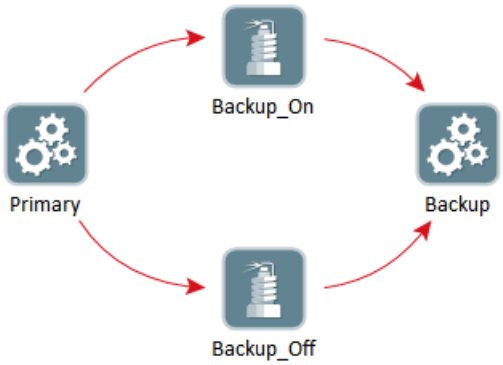
For example, if we were simulating an automobile, we could model (in great detail) the accumulated mileage (accounting for seasonal trends, etc.). We would then define this as the FMCV for various failure modes:



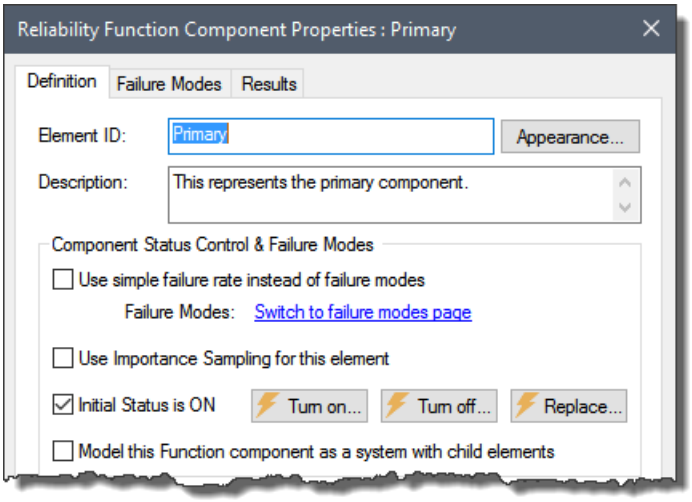
Standby Systems

Many systems have backup components that can be switched on in the event of a failure of a primary component. Such a system provides an excellent example of the power and flexibility of GoldSim, and also provides an example of the use of the ***Action*** element.

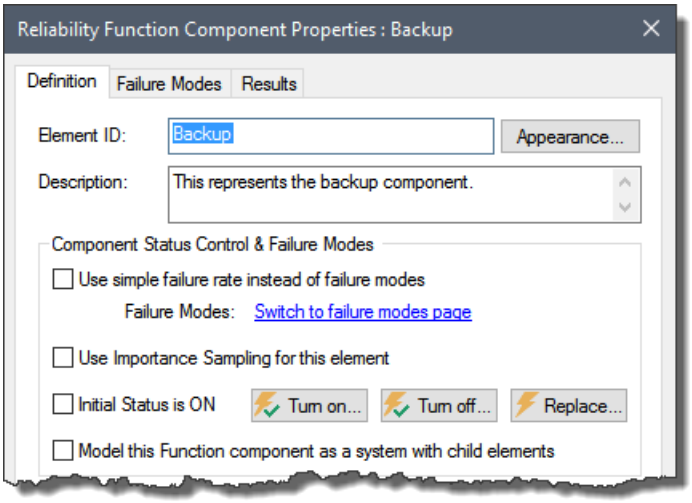
The example we will consider is shown below:



Primary and Backup are simply Function elements (like those we have discussed previously). They have identical failure modes. There is one key difference: Primary is initially On (the **Initial Status is ON** is checked):



while Backup is initially Off:



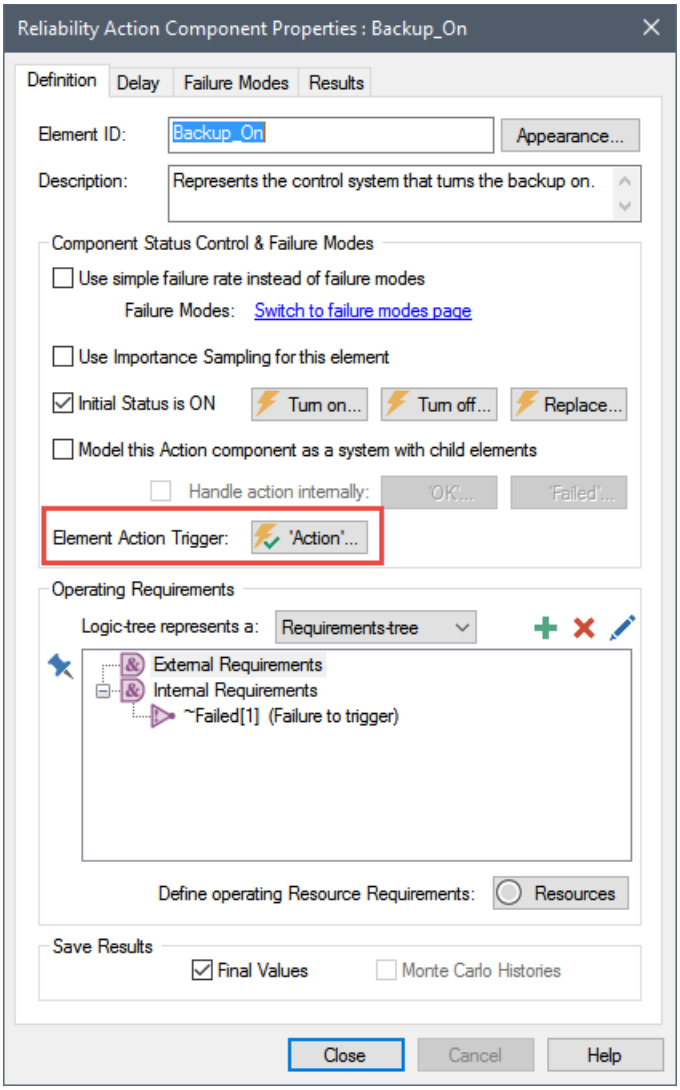
Note the **Turn on…** and **Turn off…** buttons. We will discuss these shortly.

We've already mentioned that Function elements output a Status. However, this is not their only output. Among other things, they output several types of "events". Recall from earlier in this paper we discussed how GoldSim elements can generate and process events (discrete occurrences or transactions). Whenever a Function element fails, it generates an event (named StopOperating). Whenever it is repaired, it generates another event (named StartOperating). We can then use these two events to model this system.

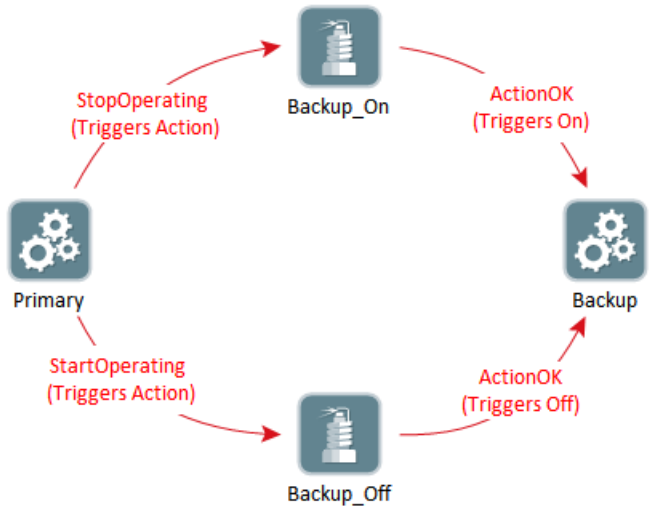
The elements named Backup_On and Backup_Off are ***Action*** elements. They are similar to Function elements (e.g., they can fail), but they are used to model different kinds of components. Whereas Function elements are used to model components which operate continuously once turned on (e.g., pumps, engines), Action elements are used to represent components which must respond to a control command or condition (e.g., switches, relays). In this example they represent switches that turn the Backup component on and off. Note, however, that these may fail when triggered to do so (i.e., they

can ***fail on demand***). In GoldSim, among other things, we can specify a probability that a triggered Action will be successful.

Action elements are triggered to act by a specified event, and if they are successful, they in turn emit an event named ActionOK. (If the Action is unsuccessful, it generates an event named ActionFailed.) The Action dialog looks like this (note the **Element Action Trigger**):



The StopOperating event from the Primary is the **Element Action Trigger** for Backup_On. It, in turn, if successful, emits an event (ActionOK) that triggers the Backup to turn on (via the **Turn on…** button in the Function dialog). Once the Primary is repaired, it emits a StartOperating event that becomes the **Element Action Trigger** for Backup_Off. It, in turn, if successful, emits an event (ActionOK) that triggers the Backup to turn off (via the **Turn off…** button in the Function dialog). So the annotated logical structure looks like this:



This provides a powerful, intuitive and flexible way to model standby systems (and failure on demand).

Non-Fatal Failures

Sometimes components can fail in such a way that the system still operates, but does not operate optimally or as designed. An example of this is the failure of three state devices. Three state devices are components that can be unfailed, can fail "open" or can fail "closed" (shorted).

The previous discussion should provide an indication of how such a device can easily be represented in GoldSim. An Action element would be used to represent the device (e.g., a switch or valve) that "opens" or "closes". Based on whether or not the Action is successful when triggered, various events are generated (ActionOK or ActionFailed), and by appropriately responding to these events, GoldSim can then track at any given time the state of the device.

A more interesting (and complex) example of a non-fatal failure is a case where a failure causes degraded performance. Imagine, for example, a pump that normally pumps at a particular speed (and hence has a particular outflow). Perhaps one of its failure modes results in the entire pump stopping. But perhaps another failure mode may simply cause the pump to operate at a slower speed (resulting in a lower outflow). How would we model that? We will discuss that very important topic in the next section.

Modeling Consequences of Failure (System Performance)

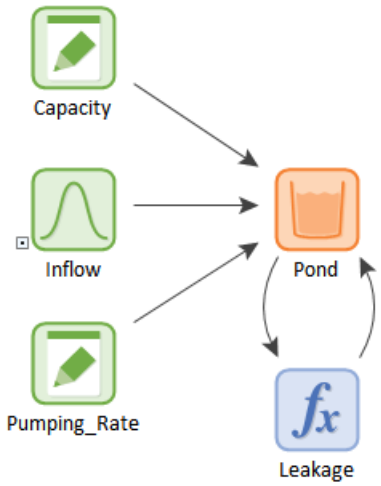
In the previous sections we have provided an overview of the power and flexibility that GoldSim provides for modeling the failure and repair of components in both simple and complex systems (and computing reliability metrics such as reliability and availability, as well as carrying out causal analysis).

However, although these metrics and analysis can be of value and interest, what is often of greater interest are the actual ***consequences*** of failure (e.g., changes in throughput, costs, and other measures of system performance). That is, the entire reason we are modeling the reliability of the system in the first place is because it performs some function (e.g., moves/processes material) , and we want to optimize key measures of that function (e.g., the throughput of material, the unit cost of processing the material).

This is easily facilitated within GoldSim because it is first and foremost a powerful and extremely flexible general purpose probabilistic dynamic simulator that has been used to simulate the behavior and evolution of a wide variety of complex systems ranging from environmental systems (e.g., mines, watersheds, waste disposal sites) to engineered systems (e.g., processing facilities, machines, space missions) to business systems (e.g., companies, projects). That is, GoldSim has the capability to realistically model the performance of complex systems.

By combining these fundamental capabilities with the features we have described above (modeling the failure and repair of engineered components), GoldSim makes it possible to build "total system models" that can represent 1) ***evolving environmental conditions***; 2) the realistic, ***dynamic complexity of failure*** of components within the system (e.g., complex interdependencies, failure rates that respond to evolving environmental conditions); and 3) the actual ***consequences*** of failure (e.g., changes in throughput, costs, and other measures of system performance).

To illustrate this in a very simple example, let's consider the case of the pump discussed above. We will do this by revisiting the simple pond model that we discussed at the beginning of this paper. Recall that water flowed into the pond, the pond leaked, and a pump removed water from the pond. The model looked like this:

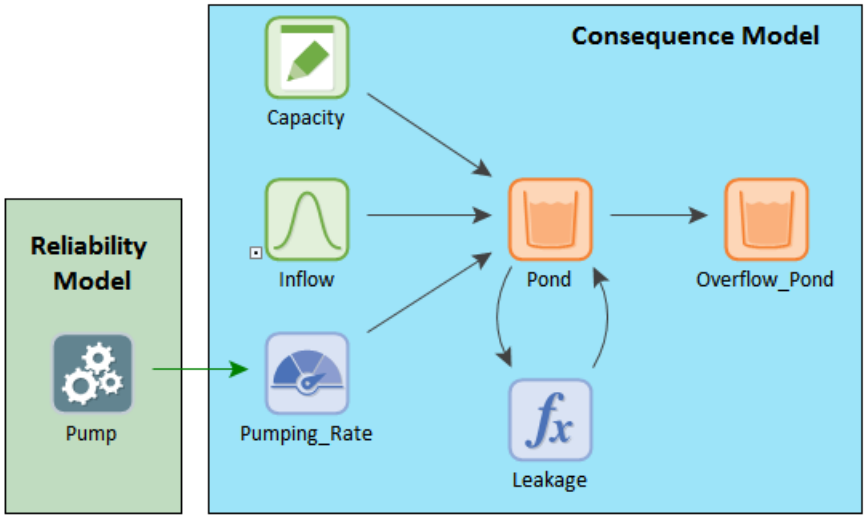


The pond had a capacity, but in our example, this was never reached. We will modify this simple model in two ways:

1. We will assume that the pump can fail with two different failure modes:
   - One shuts the pump down completely;
   - One cuts the pumping rate in half (from 5 m3/day to 2.5 m3/day)
2. When the pond reaches its Capacity (175 m3), it overflows into a second pond.

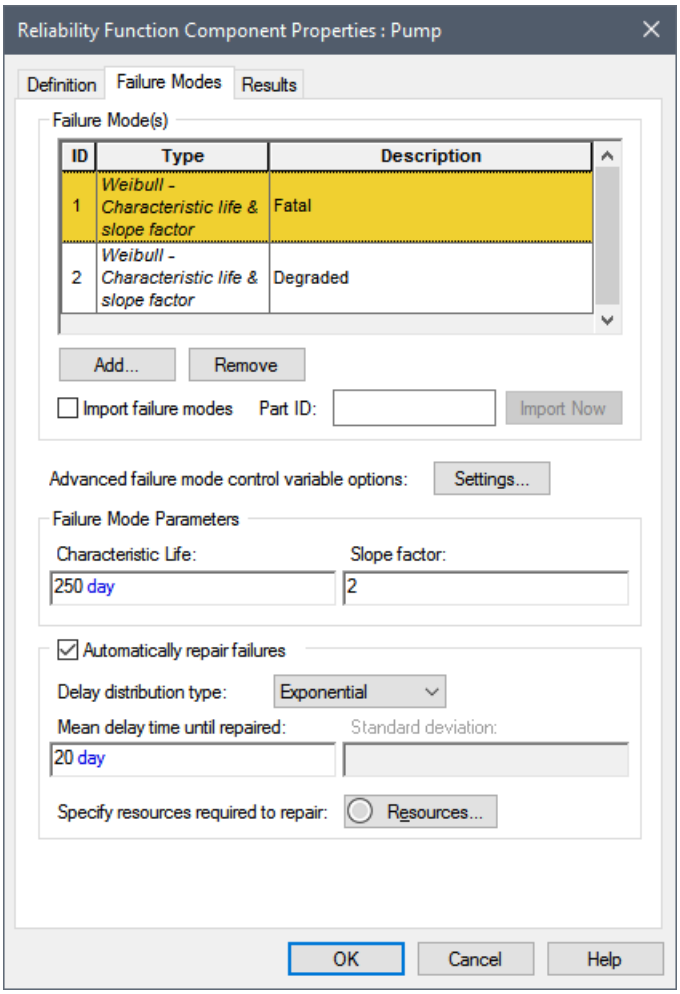
We are interested in how much water overflows over the period of interest (say 1 year). That is, our performance measure (i.e., the consequence that we are interested in) is the cumulative amount of water that flows into the overflow pond over the year.

The new model structure to represent this looks like this:

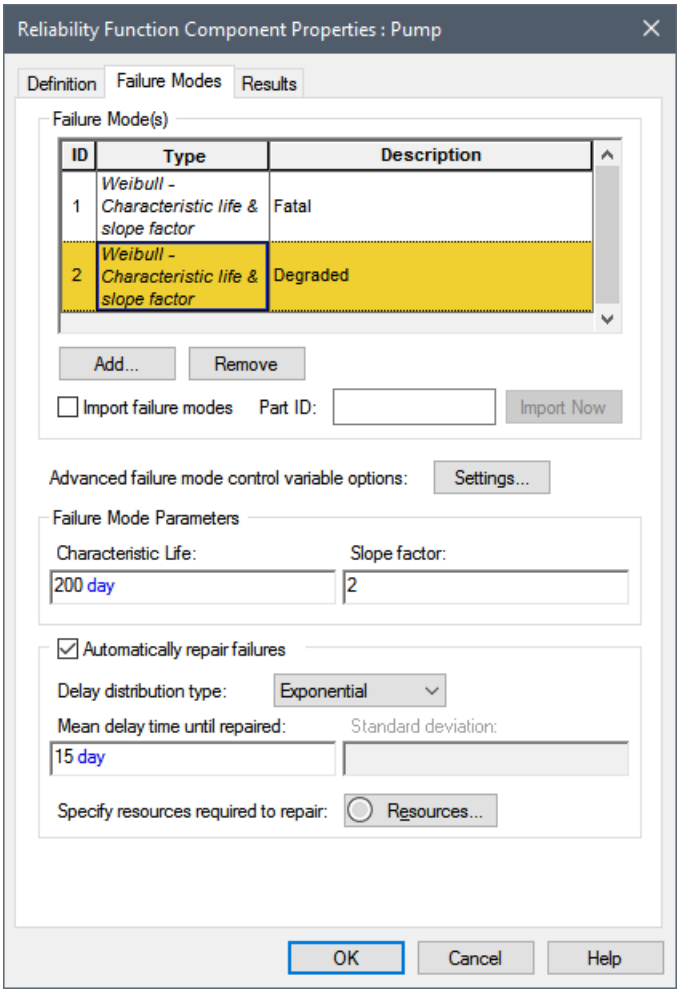


The Pump element has two failure modes defined. Both are Weibulls.

The first failure mode has a Characteristic life of 250 days and takes approximately 20 days to repair:

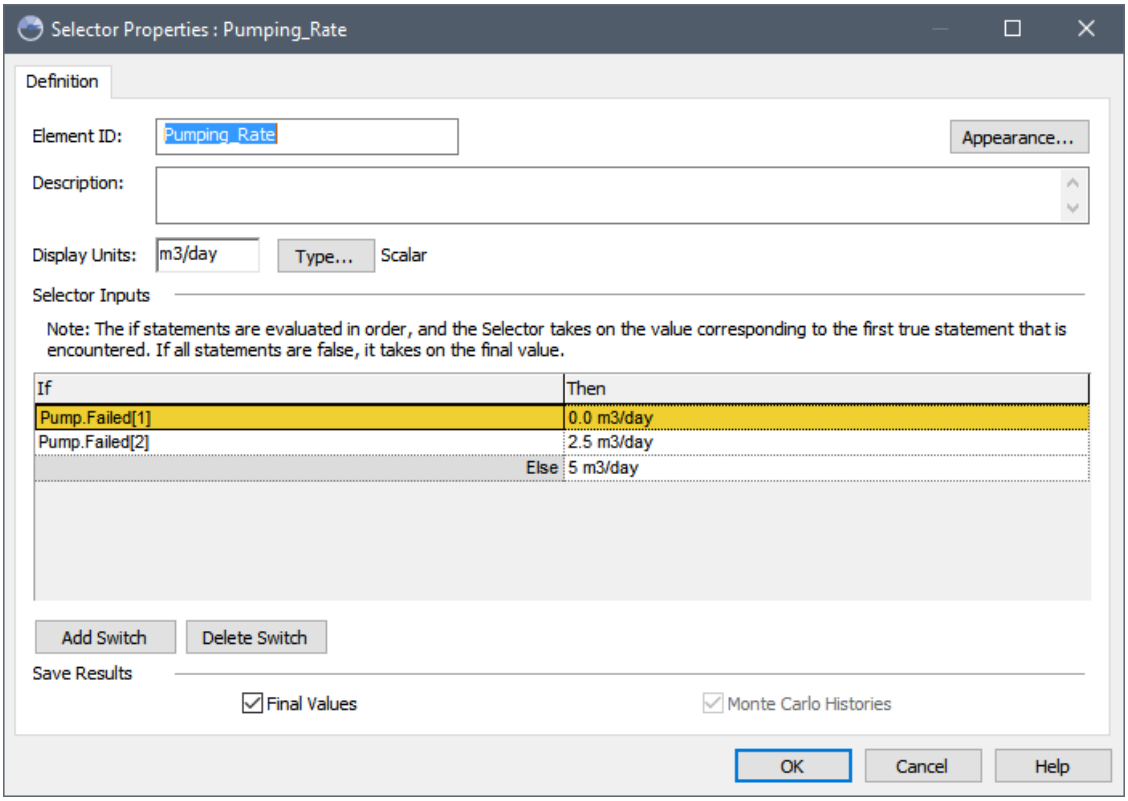


The second failure mode has a Characteristic life of 200 days and takes approximately 15 days to repair:



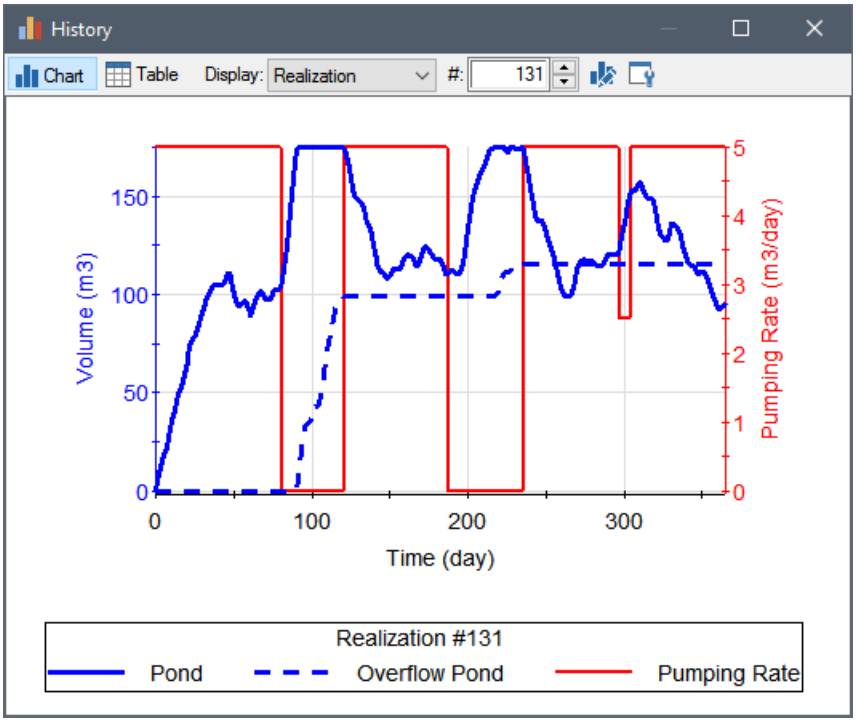
In addition to representing the failures and repairs, however, we want to represent the consequences of the failures. In particular, we want to represent that fact that if the pump fails by the first mode, it is fatal (it stops pumping completely), while if it fails by the second mode, the pumping rate is cut by half. So how do we represent these consequences on the pumping rate?

We've mentioned several times that Function (and Action) elements have multiple outputs that can be referenced (e.g., Status, StopOperating, StartOperating). Another of these outputs identifies whether or not the component is currently failed by a particular mode. In this case, an output named Pump.Failed[1] is true if the pump is currently failed due to the first failure mode, and false otherwise. Similarly, an output named Pump.Failed[2] is true if the pump is currently failed due to the second failure mode, and false otherwise. These can then be used to define the Pumping_Rate as a function of time. The Pumping_Rate element is what is known in GoldSim as a ***Selector*** element. A Selector simply provides a straightforward way to create nested if, then logic:



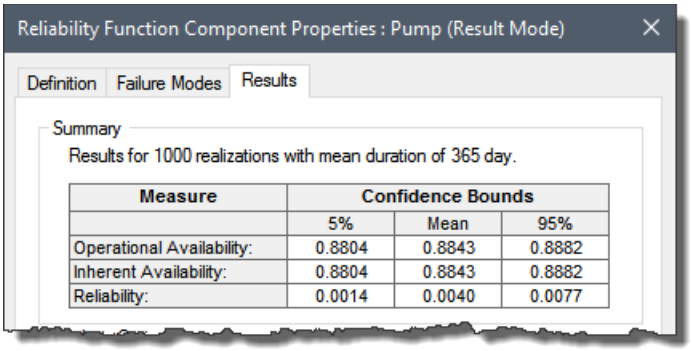
As a result of this, the Pumping_Rate dynamically changes during a simulation as a result of failures and repairs.

We can see this if we run the model (for 1 year and 1000 realizations), and view a time history result. This plot shows the volume in the two ponds, as well as the pumping rate, for a single realization (#131):



In this particular realization, the pump fails twice due to the first mode (such that the pumping rate goes to zero) and once due to the second mode (such that the pumping rate drops to 2.5 m3/day). In all three cases, this causes the water volume to increase.

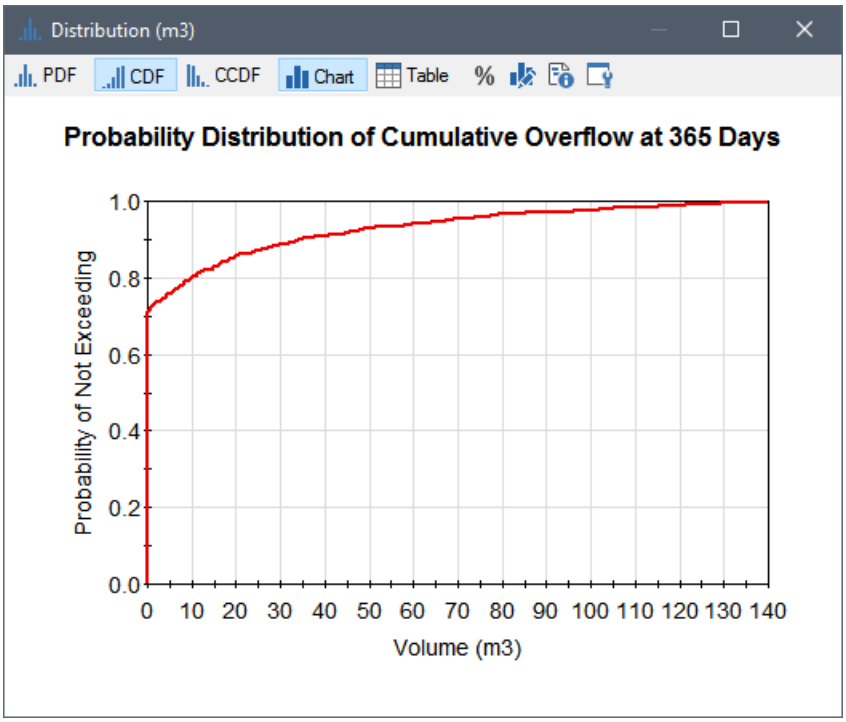
Of course, we can compile all of the failures for all 1000 realizations to view standard reliability metrics for the pump:



Measure	Confidence Bounds		
	5%	Mean	95%
Operational Availability:	0.8804	0.8843	0.8882
Inherent Availability:	0.8804	0.8843	0.8882
Reliability:	0.0014	0.0040	0.0077

We see that the Reliability of the pump is close to zero (i.e., it almost never operates for the entire year) and the Availability is about 88% (i.e., it is operating 88% of the time).

Although this may be of some interest, it is not what we really what we want to know, as it says nothing about the consequences of the failure. The consequence of failure is that the pond can *potentially* overflow (not all failures result in an overflow, depending on the inflow, the failure mode, and how quickly the pump is repaired). Because we built a consequence model, however, we can readily quantify this. Here is the probability distribution of the cumulative overflow from the pond:



This indicates that there is about a 30% chance of the pond overflowing. If it does, the cumulative overflow could be as high as about 130 m3.

Here we see the true power of a simulation-based reliability modeling approach: the ability to predict the consequences of failure. In fact, one could argue that this is the entire objective of reliability modeling. By tying predicted failures to predicted consequences, we can use modeling results to make design decisions. For example, in this case, if the predicted consequence was unacceptable (e.g., if it would result in exceeding a regulation), we could change the design of the system (e.g., have a backup pump available).

GoldSim's Approach to Probabilistic Risk Assessment

We've just described in some detail how GoldSim can be used for reliability modeling. Reliability modeling and probabilistic risk assessment (PRA) share many common features, since they both deal with failure of various components and systems. As a result, much of what has just been discussed for reliability modeling is also applicable for probabilistic risk assessment.

However, these two types of analyses traditionally use different types of approaches, since they are typically focused on different types of results. Reliability models focus on computing the reliability and availability of a system, and are typically used to compare design (including preventive maintenance) alternatives with the goal of optimizing things like throughput, warranty and/or maintenance costs.

Probabilistic risk assessment, on the other hand, was initially developed to analyze systems such as nuclear power plants and space missions, in which the consequence of failure is very high (e.g., can lead to injury, loss of life, severe damage to the system, or perhaps damage to the surrounding environment). Hence, it focuses on predicting the probability of those events that lead to such consequences. For these kinds of systems, due to the nature of the consequences of failure, backup and redundancy are often a key part of the design, such that a failure is usually caused by a (presumably rare) combination of events.

That is, in a PRA, 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. The ultimate goal is not to optimize things like throughput or costs, but to evaluate system safety and inform design or operational changes that can minimize the probability of such failures.

As a result, the conventional approach to risk assessment 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. This is typically done through logic-based procedures (i.e., event trees/fault trees).

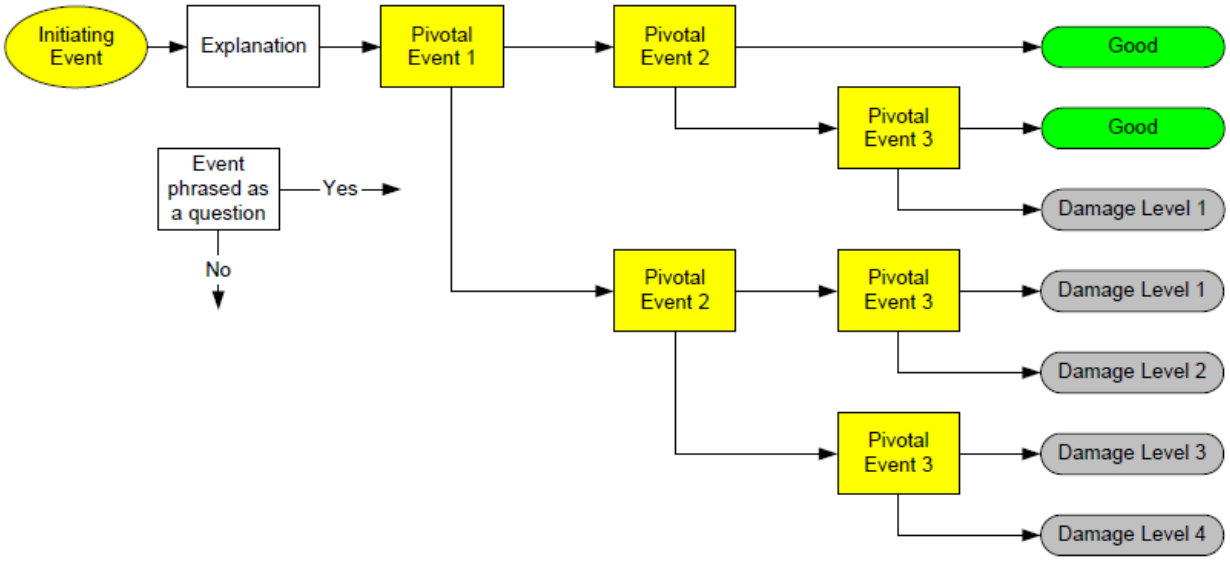
In the sections below, we briefly discuss how such analyses can be carried out in GoldSim.

Basic PRA Concepts

This section very briefly describes some basic PRA concepts, specifically focusing on traditional approaches (event trees/fault trees). It is not meant to be exhaustive and discusses only the basic concepts. The objective is simply to provide the basis for discussing how these concepts translate to a simulation-based approach. The discussion below is based on Stamatelatos et al (2011) and Vesely et al (2002).

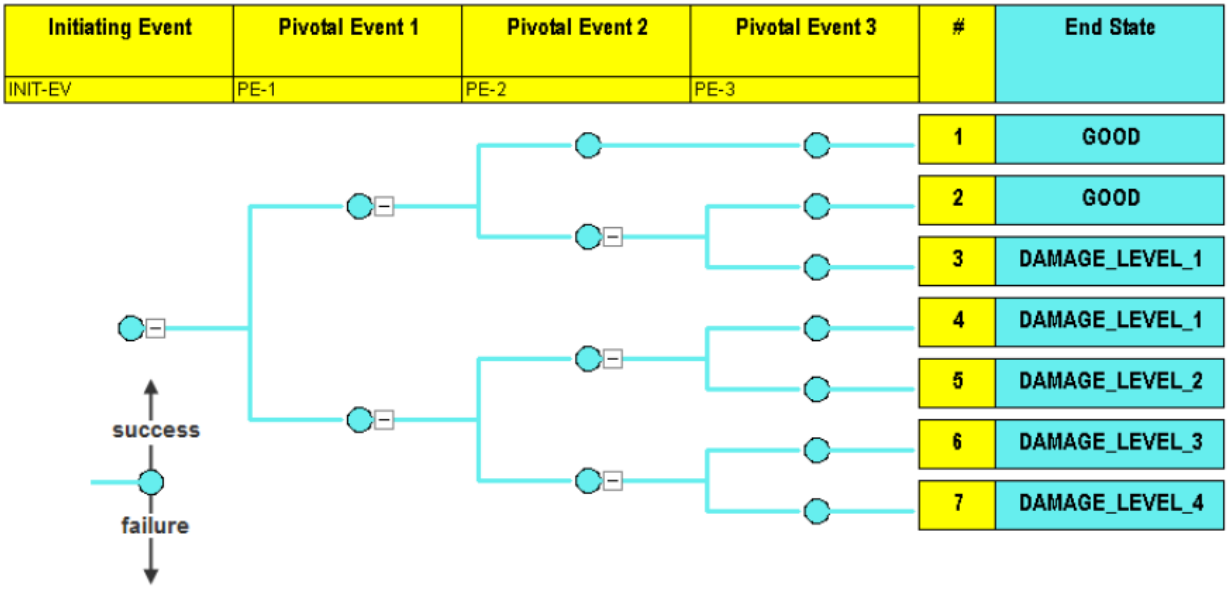
A PRA attempts to model a sequence of events that need to occur in order for undesired end states (e.g., catastrophic failures) to occur. This is done by representing a set of ***scenarios*** that have specified frequencies and consequences. A scenario starts with an ***initiating event (IE)*** that perturbs the system. This perturbation requires a response from one or more systems (or operators). The IE is followed by one or more ***pivotal events*** leading to a particular end state. The pivotal events include successes and failures of responses to the IE, as well as the occurrence (or non-occurrence) of key conditions or phenomena. The various pivotal events eventually lead to possible end states (some of which are undesired).

One way to represent the scenarios that ensue from a given IE is in the form of an ***Event Sequence Diagram (ESD)***. This is essentially a flowchart with paths leading to different end states. Each path through the flowchart represents a scenario:



Typical Structure of an Event Sequence Diagram (from Stamatelatos et al, 2011)

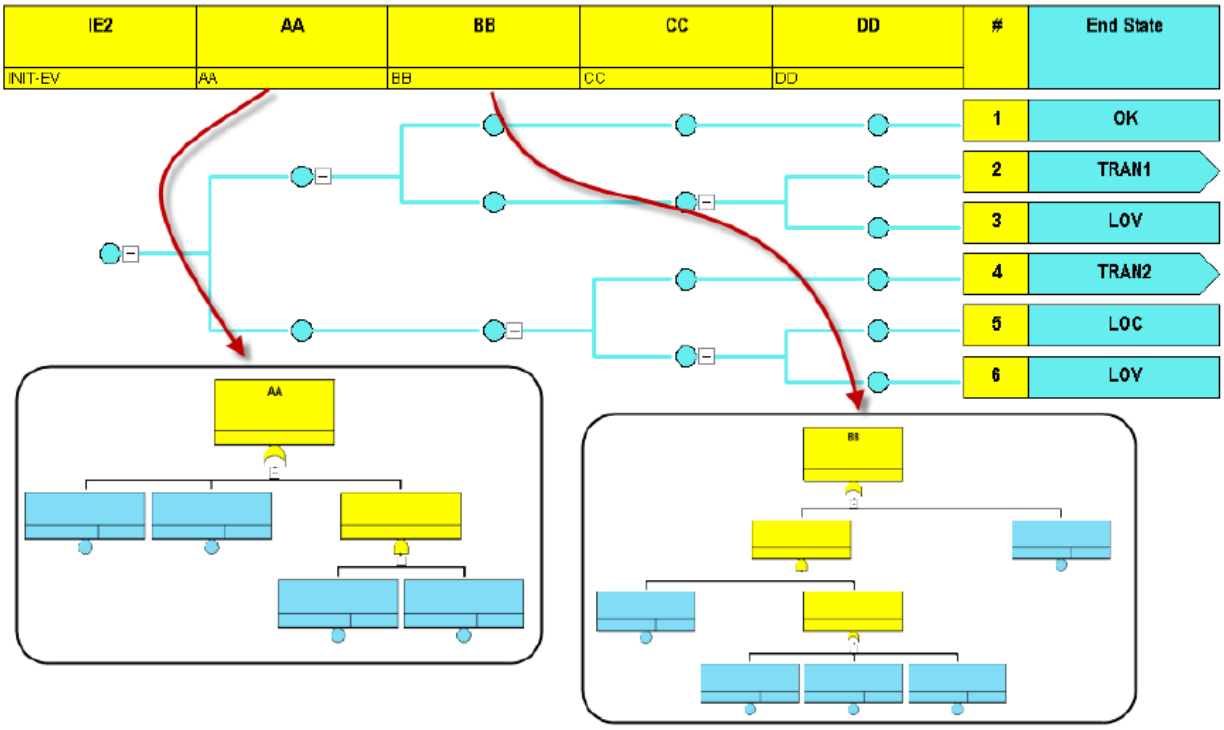
An ***Event Tree (EV)*** presents this same information in a tree structure, and facilitates a quantitative analysis:



Event Tree Representation of Event Sequence Diagram Show Above (from Stamatelatos et al, 2011)

Each path through the tree is a scenario. Given a probability for each node, each logical sequence leading to an end state can then be represented using simple logical operations, and the probability of each end state can be computed. In some cases, the probabilities for the nodes can be assigned directly

(e.g., based on experimental data). More frequently, pivotal events are represented using ***Fault Trees (FT)***. In a Fault Tree, the sequence of events leading to the occurrence of the "top event" is systematically divided into events whose probabilities can be estimated. Fault Trees are constructed using logic gates (e.g., AND and OR gates):



Schematic Illustrating How Selected Pivotal Events in an Event Tree are Represented Using Fault Trees (from Stamatelatos et al, 2011)

To carry out a traditional PRA, a realistic (and often quite large) set of scenarios must be developed and quantified in this manner.

In the sections that follow, we briefly discuss how these concepts translate into GoldSim's simulation-based approach to PRA.

Modeling Initiating Events in GoldSim

When carrying out a PRA in GoldSim, the same type of conceptual systems analysis described above is required in order to identify system components, their behavior, and initiating events (using, for example, graphical tools such as Event Sequence Diagrams). However, the way that the various events are represented and modeled is different in a simulation-based approach.

Let's begin by discussing how GoldSim represents initiating events. In a GoldSim PRA model, initiating events can be subdivided into two categories:

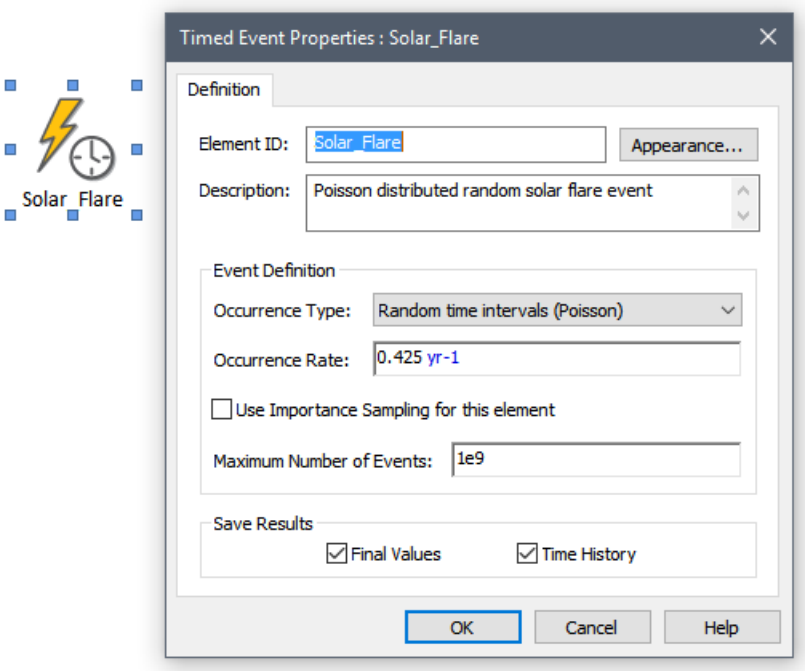
- A random external or internal event. Examples of external events include a solar flare or terrorism event. Examples of random internal events might be an unplanned human action or running out of fuel.

- The failure of one or more system components.

These two types of initiating events are treated differently in GoldSim.

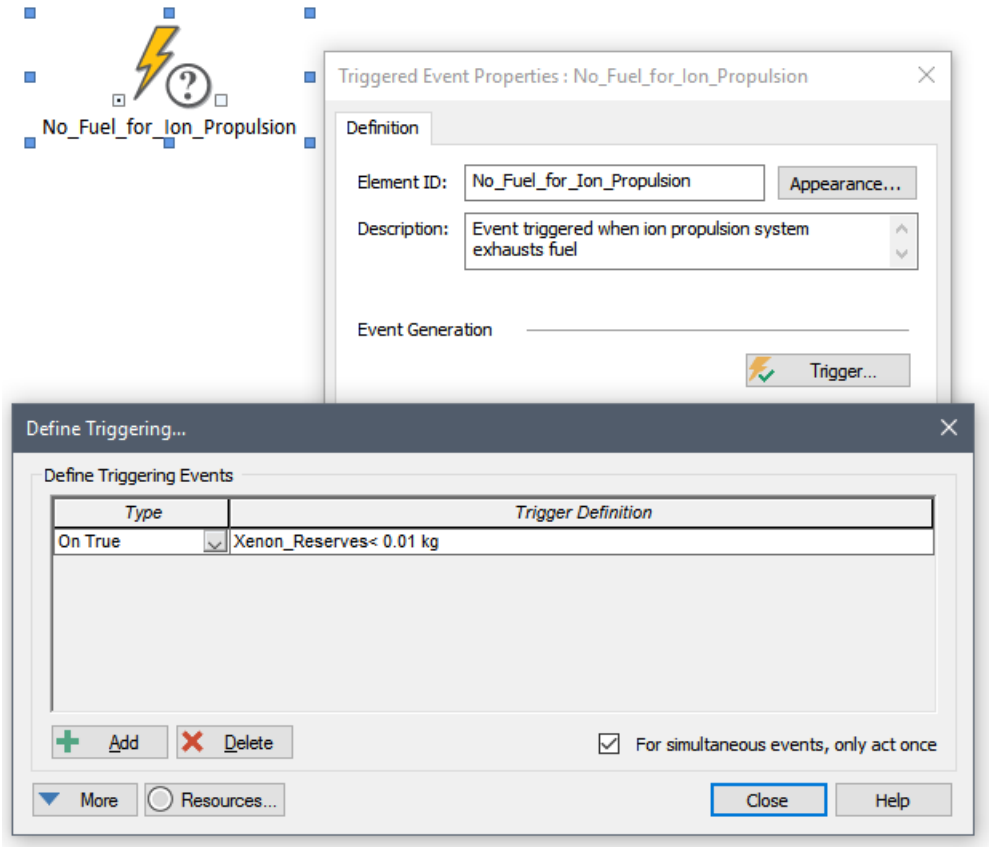
Modeling Random Initiating Events

Random (external or internal) events are modeled using ***Timed Event*** or ***Triggered Event*** elements. A Timed Event element randomly generates events based on a specified rate:



Note: Although by default Timed Event elements represent random events as Poisson processes (i.e., the time intervals between events are exponentially distributed), any distribution can be selected to represent the time intervals between events..

Triggered Event elements can represent events that are trigged by circumstances (running out of fuel):



Of course, the great power of a simulation-based approach in this regard is that the model itself can simulate the evolution of the condition(s) leading to the event (in this example, the amount of fuel remaining).

Modeling Initiating Events Resulting from Failures

In some cases, the initiating event is not a random external or internal event, but simply a (presumably rare) failure of a component or system of components.

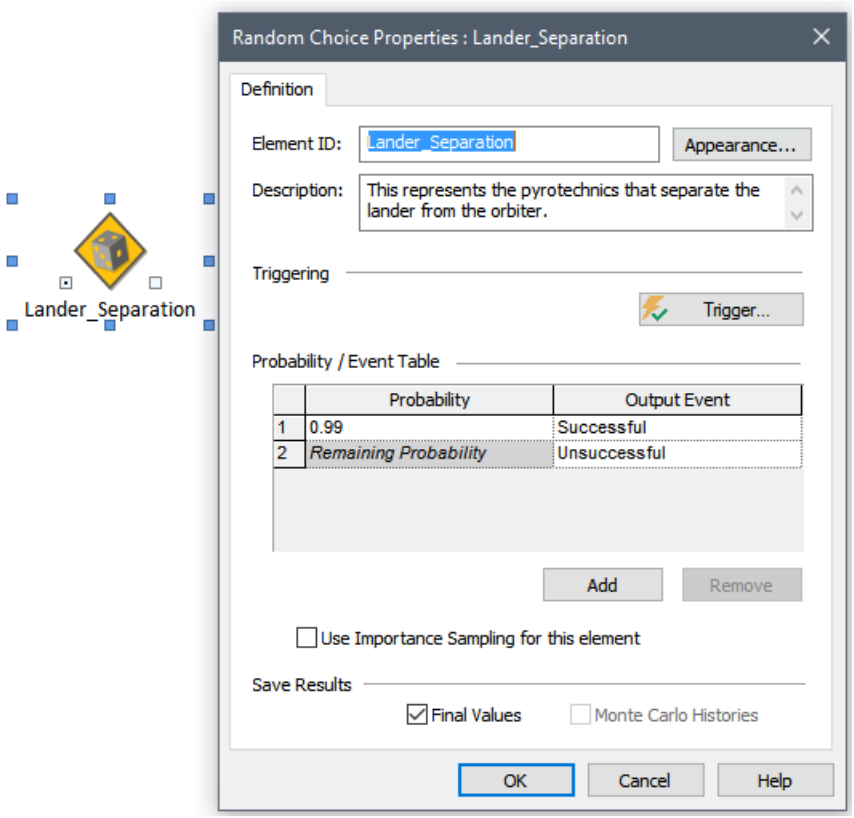
We have discussed in detail in the first part of this paper the powerful and flexible capabilities that GoldSim has for realistically modeling failure events using Reliability elements (Function elements and Action elements). These features can, of course, be used to model complex failure scenarios that could act as an initiating event.

Modeling Pivotal Event Sequences in GoldSim

In conventional PRA analyses, the potential effects of an initiating event are represented using an Event Tree, with the consequences of an initiating event cascading through a series of chance nodes (the ***pivotal events***). The outcome of each pivotal event reflects the probabilistic state of a particular component of the system (e.g., does the detector detect the smoke?). These tree-based approaches rely on the judgment of analysts who have the experience and imagination to identify all potentially-significant event sequences.

In the simulation model, however, the user has only to define the elements that are directly affected by an event. The effects of an initiating event (the various sequences of events) then arise naturally out of the model's logic, as the elements that are affected respond to the event and its consequences propagate through the model. This is another way in which the simulation approach is distinguished from traditional tree-based approaches. In a sense, the simulation approach "discovers" failure scenarios, as opposed to the classical PRA approach where the analyst has to define all of the failure scenarios up front.

Within GoldSim, two quite different approaches can be taken to represent a pivotal event. The simpler of these is to add a ***Random Choice*** element, which has a set of user-defined outcomes with associated probabilities:



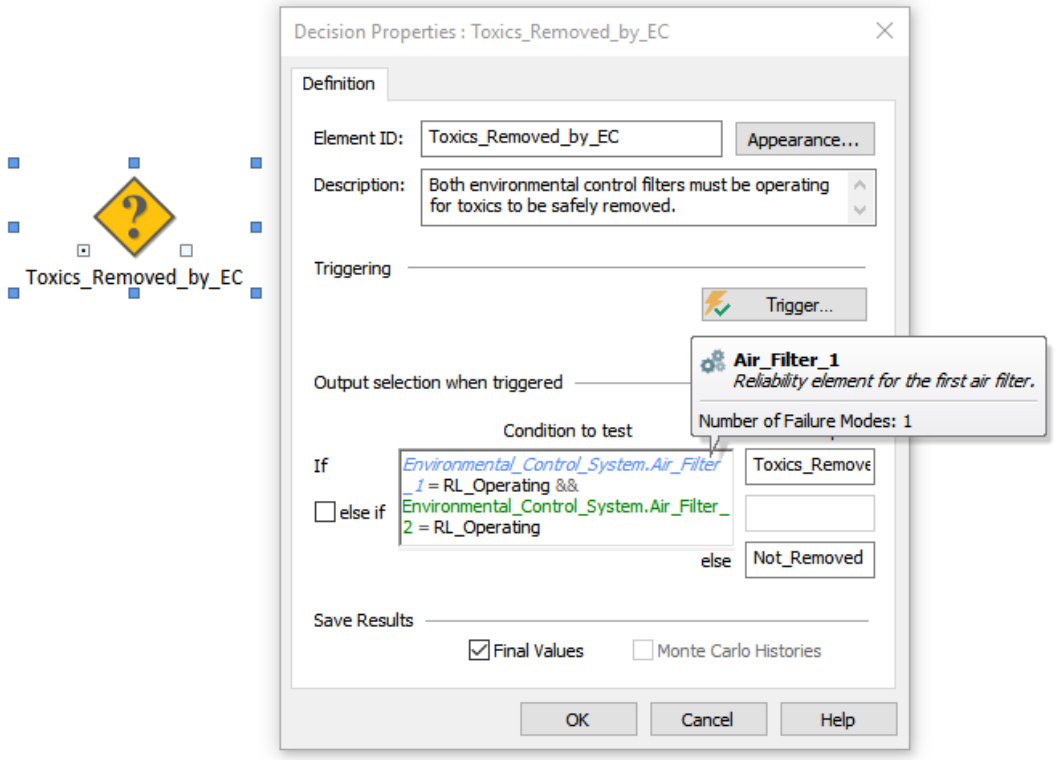
Each outcome is associated with a different element output. Upon receiving notice of a triggering event, the Random Choice element "rolls the dice" to randomly select which outcome should occur, and emits an event from the associated output.

This approach is simple, but may not be realistic. When an initiating event sets a train of pivotal events in motion, the outcome usually depends on the states of some physical components of the system. If a component is not operating when required, negative outcomes may ensue. In the simplest case, for a non-repairable component that has only operating and failed states, with a constant hazard (failure) rate, the probability that the component is operating when the event occurs simply equals its projected reliability. This can be readily represented using the Random Choice element.

However, if the failure rate is not constant, or if the component has more complex failure, repair, switching, or maintenance behaviors, such a simple approximation may not be adequate. For example, what if the component is normally repaired when it fails, but the necessary spare part may not be available? What if its aging rate depends on its operating environment?

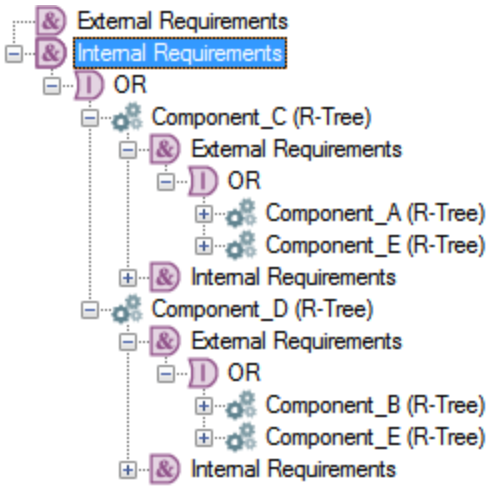
An alternative approach for representing pivotal event sequences that can handle these kinds of complexities is to use a Reliability element and simulate its state dynamically. When the precedent event occurs, the state of the component or system represented by the Reliability element is known within the simulation (e.g., it may be operating normally, or failed, or undergoing maintenance or repair, or inoperable because of a missing requirement, etc.). The Monte Carlo simulation process effectively samples these possibilities as it cycles through a number of realizations.

Note that within a GoldSim model, the current status of a Reliability element can be readily queried using a ***Decision*** element:

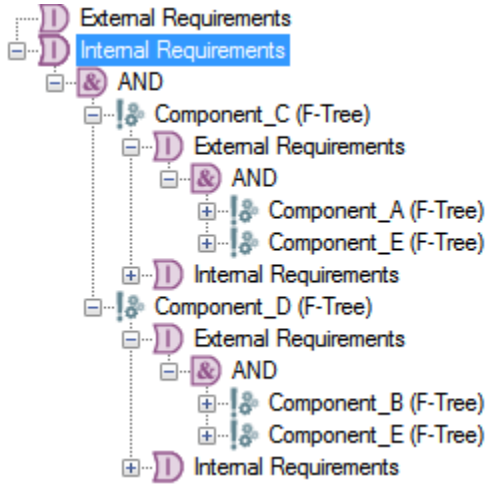


In this example, the Decision element queries the status of two Reliability elements to determine if they are operating successfully. Based on this, it emits one of two possible events (in this case "Toxics_Removed" or "Not_Removed") that then trigger other elements in the sequence to determine the relevant end state.

Recall from our discussion above that pivotal events are often modeled using Fault Trees. As we have seen, Reliability elements themselves usually incorporate complex Fault Tree logic. In the discussions earlier in this paper, Operating Requirements for Reliability elements were presented in terms of Requirements trees:



However, a Requirements tree can also be defined and interchangeably viewed as a Fault tree (in which, for example, AND gates become OR gates, and vice versa):



Compared to the use of a simple Random Choice element, the use of Reliability elements to represent pivotal event sequences requires much more input data, and is more complex computationally. Hence, the modeler has to choose between the simplicity of a Random Choice element and the realism of a Reliability element. Typically, Random Choice elements will be used in preliminary versions of a model, and replaced by Reliability elements in later versions.

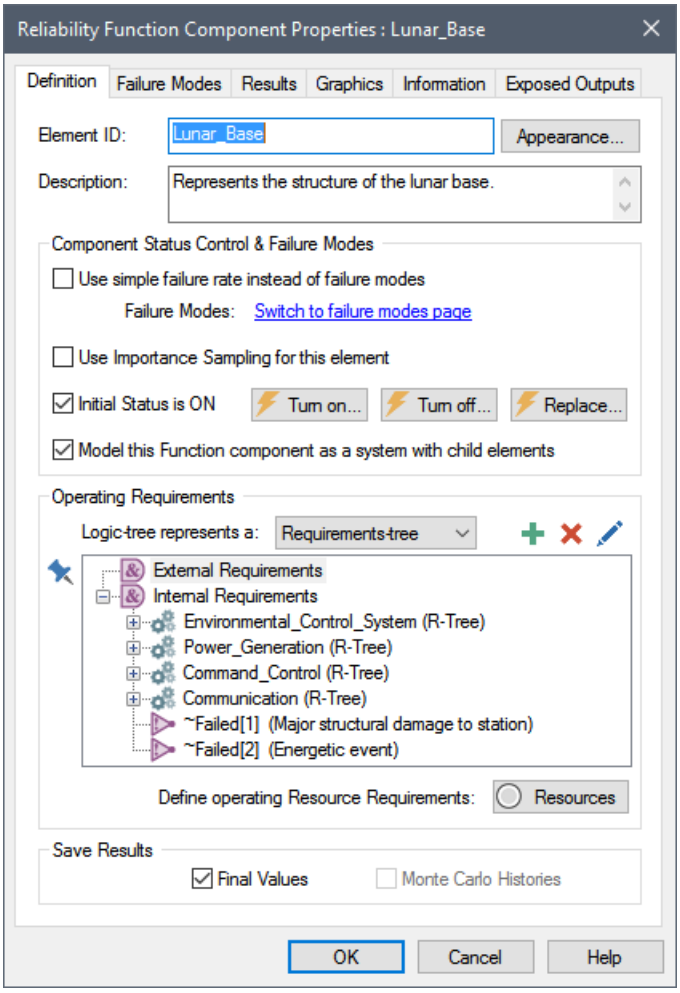
Example PRA GoldSim Applications

The brief discussion presented above describes how key PRA concepts translate into a simulation-based approach. To illustrate this further, two simplified PRA applications will be briefly discussed. These two aerospace applications are based on example models in Appendix D of Stamatelatos et al (2011). The applications will not be discussed in detail. Rather, several representative portions of the models will be discussed to highlight key features of the GoldSim simulation-based approach. (Both applications are available for download from the GoldSim Model Library, which will be discussed at the end of this paper.)

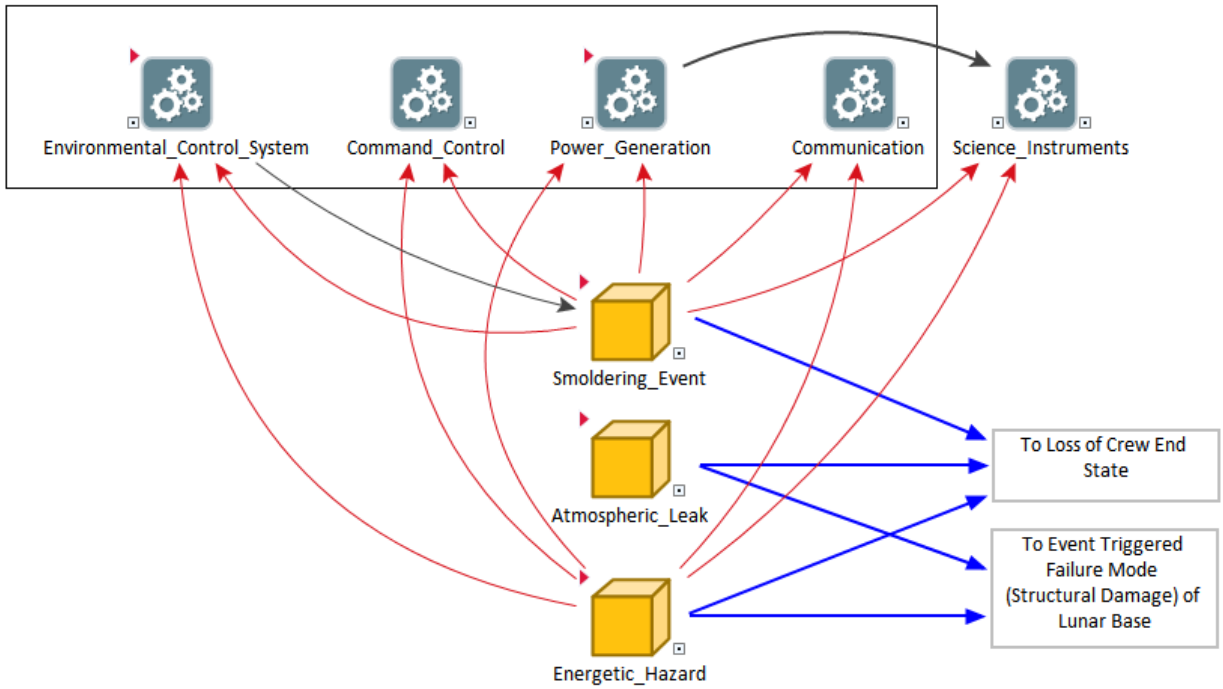
PRA of a Lunar Base

This model evaluates the performance of a lunar base over its 20 year scientific mission. The purpose of the model is to determine the probability that the mission will successfully achieve all the mission goals, along with the probability of two undesirable end states (Loss of Mission and Loss of Crew).

The Lunar Base itself is represented using a Reliability element (a Function) that is modeled as a System (a Container):



The contents of this element are shown below:

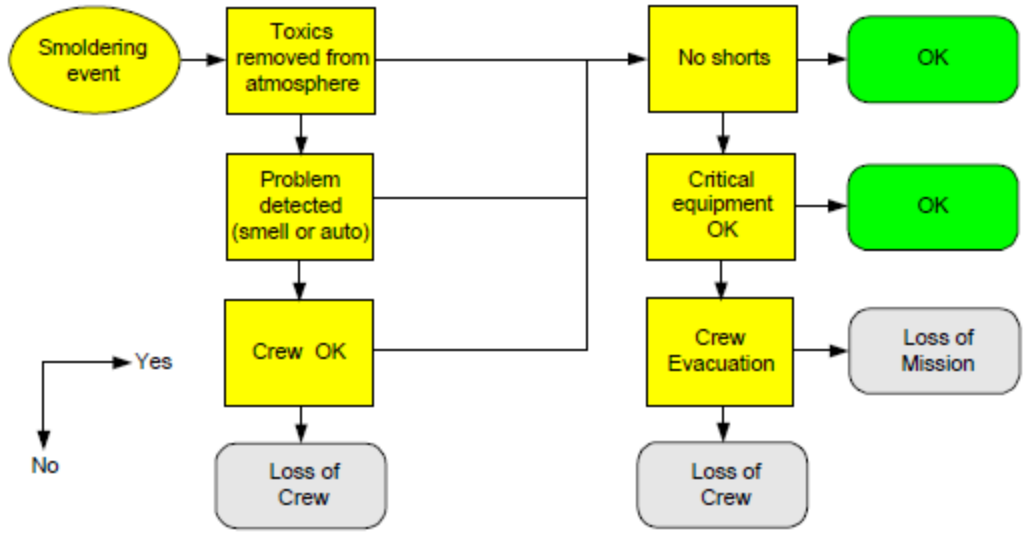


The safe operation of the base is dependent on four major subsystems: Environmental Control, Power Generation, Command and Control, and Communication (each of which is modeled as a Function element). The base's dependence on these subsystems is modeled using a requirements tree (shown in the dialog for the Lunar Base element above). Its structural integrity is modeled as failure modes that can be triggered by certain initiating events.

If any of these systems fails, or if the structural integrity of the base is compromised, evacuation is required. Evacuation is modeled by another Reliability element (an Action) triggered by the failure of the base, or the consequences of certain initiating events. The mission is always lost if evacuation is required, and if the evacuation fails it leads to Loss of Crew.

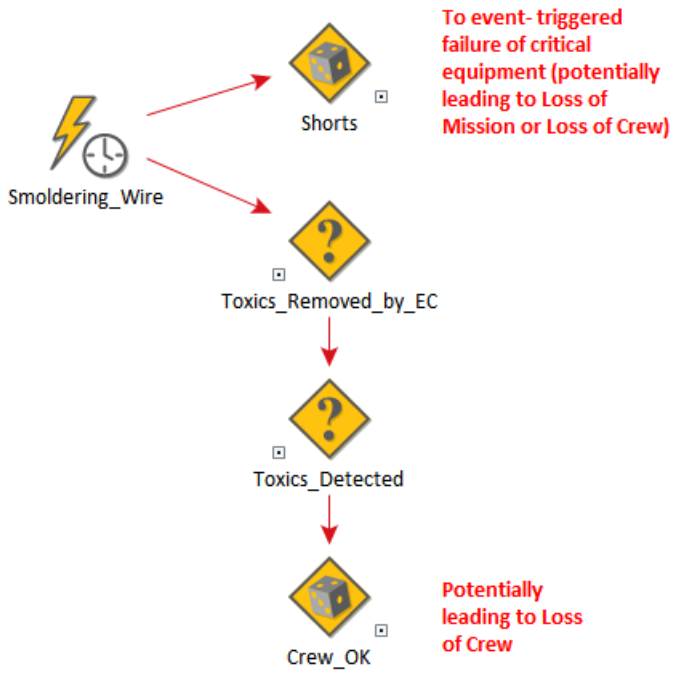
The scientific mission of the base is dependent on a number of instruments (represented above using another Function element), some of which can be replaced during a periodic resupply mission, and others which cannot be replaced. If the irreplaceable scientific instruments are destroyed, it leads to Loss of Mission.

There are a number of initiating events considered in the model. Let's look at one to get an appreciation for how such an event sequence is modeled in GoldSim. This particular sequence involves a smoldering event that results in the creation of gaseous toxics. The Event Sequence Diagram associated with this initiating event is shown below:



Event Sequence Diagram For Smoldering Event at Lunar Base (from Stamatelatos et al, 2011)

The model for this in GoldSim is shown below:



The initiating event is represented using a ***Timed Event*** element and is modeled as a Poisson process.

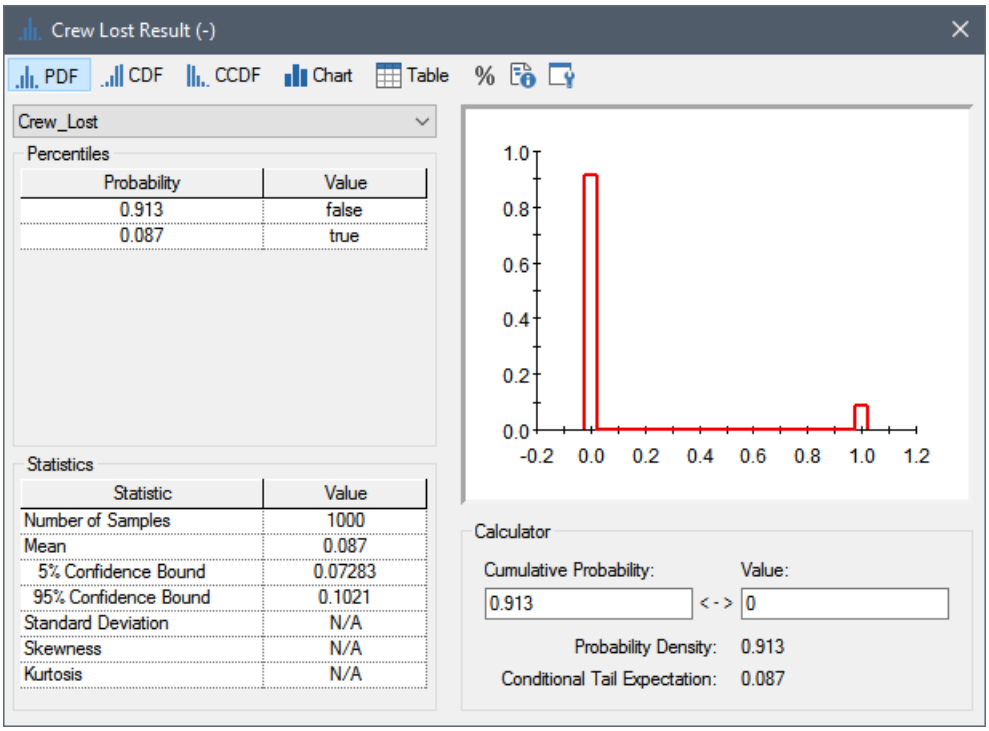
The event has two direct consequences: 1) release of toxic fumes into the base atmosphere; and 2) possible generation of electrical shorts. In contrast to the ESD approach, in GoldSim both potential consequences are immediately triggered. With regard to the toxic fumes, the current status of the filtration system is queried using a ***Decision*** element (Toxics_Removed_by_EC) that queries Reliability elements in the Environmental Control system. If these elements indicate that the system is fully functional, the toxics will be removed.

However, if the system is partially failed (in a compromised state but functional enough to safely support the astronauts), the toxics will not be removed, and GoldSim will proceed to the next Decision element (Toxics_Detected), which queries a Reliability element representing an automatic toxic detection system. If it is operating, the crew will be notified and will take action to remove the toxics. If not, GoldSim proceeds to a ***Random Choice*** element (Crew_OK) that determines whether the crew is able to detect the presence of toxics by smell. If they do, they can take action to deal with the problem. If they do not, it leads to Loss of Crew.

The electrical shorts consequence uses another Random Choice element (Shorts) to determine whether or not electrical shorts occur. If they do, there is a probability that the shorts could trigger a failure of one or more major base systems. If failure does occur, this in turn triggers the Reliability element representing evacuation, and depending on its success this leads to either Loss of Mission or Loss of Crew.

The point here is that this event sequence can be represented with dynamic realism using a combination of the powerful elements provided by GoldSim (in this case, Timed Events, Random Choices, Decisions, and Reliability elements).

The form of the key results for such a simulation look like this (in this case, the probability of Loss of Crew):

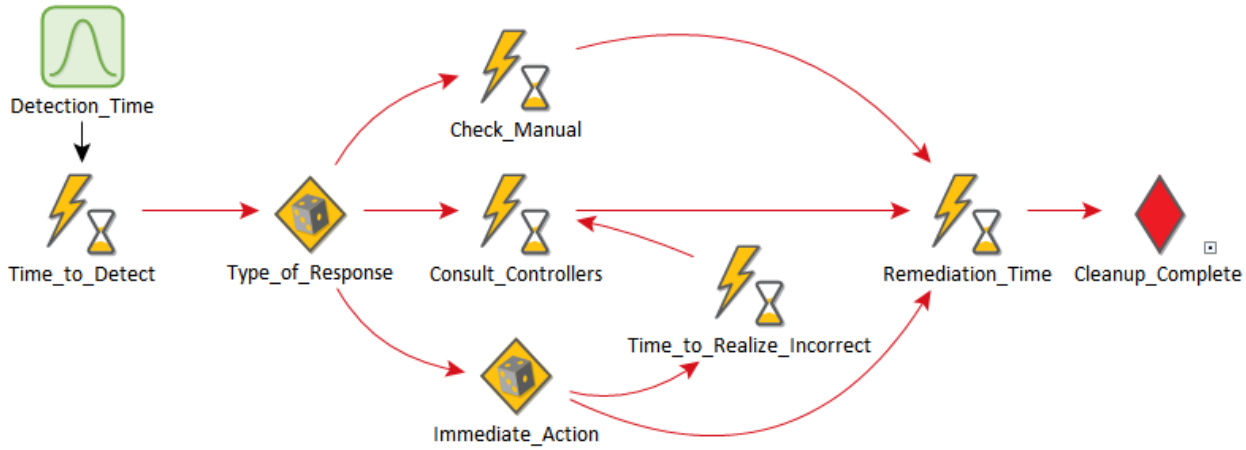


This indicates a probability of about 8.7%, with 5/95% confidence bounds (based on the number of realizations) of between 7.3% and 10.2%.

One other interesting aspect of this example model is that it provides an illustration of how the simulation approach allows for the straightforward representation of the effects of human behavior in a

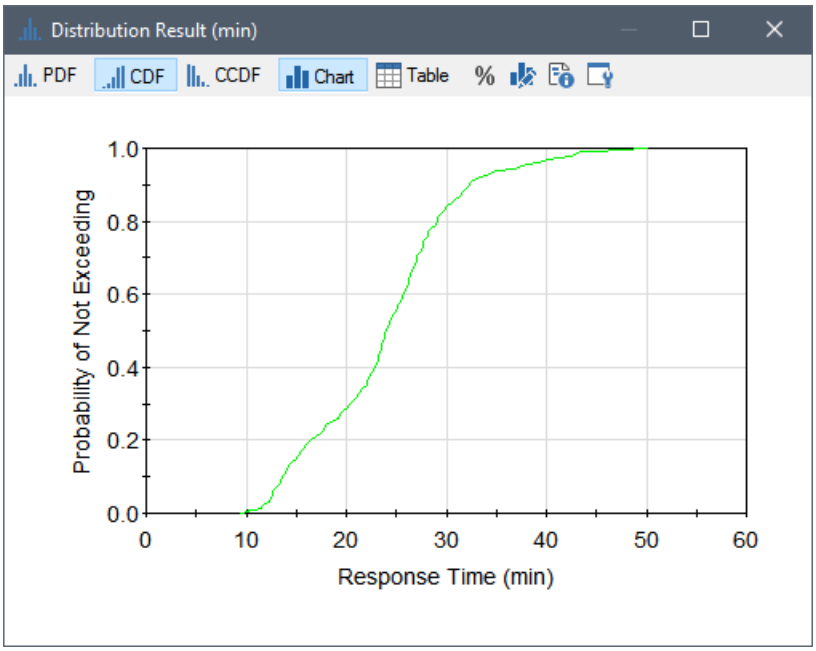
PRA. In particular, a scenario was added where crew action may be required in order to react to and repair a rupture of one of the two electrical storage battery systems.

In this scenario, the crew is required to notice and respond to the failure, taking appropriate action to mitigate the effects of any electrolyte release before irreparable damage is done to the scientific instruments or to the other battery system, either of which would result in loss of the mission. However, there is a possibility of the crew responding in an inappropriate way, so that the damage occurs even though the response was timely. A dynamic model (randomly) represents the crew's response to a leak:



In this diagram, the objects such as Time_to_Detect are ***Event Delays*** that are used here to represent processes that have a specified (probabilistic) duration to complete.

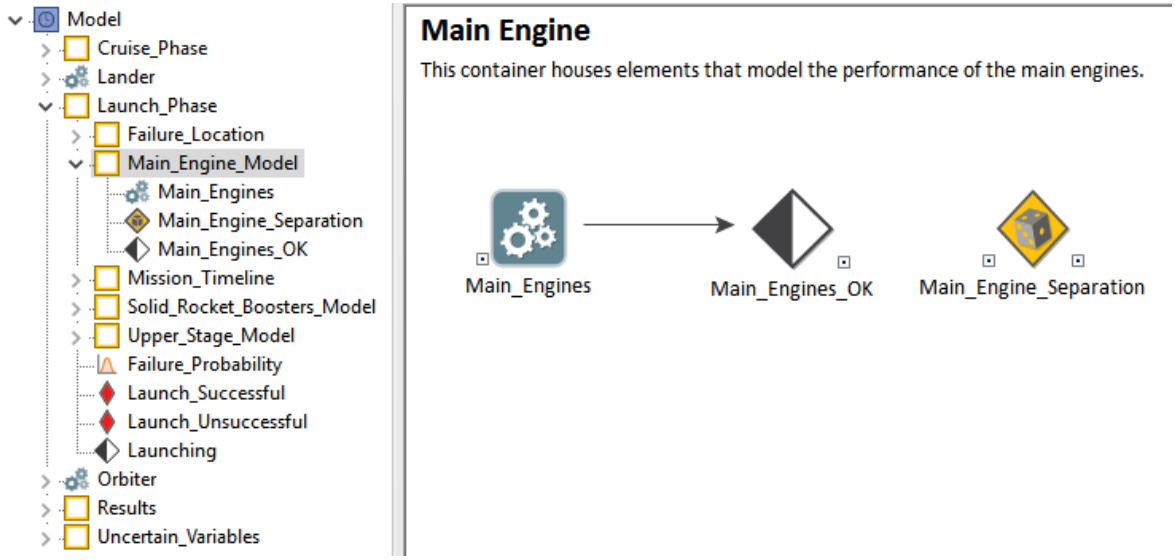
Based on this model, if the initiating event occurs, a response time is calculated, and the probability of damage to the scientific equipment is then computed as a function of the response time. Here, for example, is the simulated distribution of response times:



PRA of an Unmanned Exploration Mission

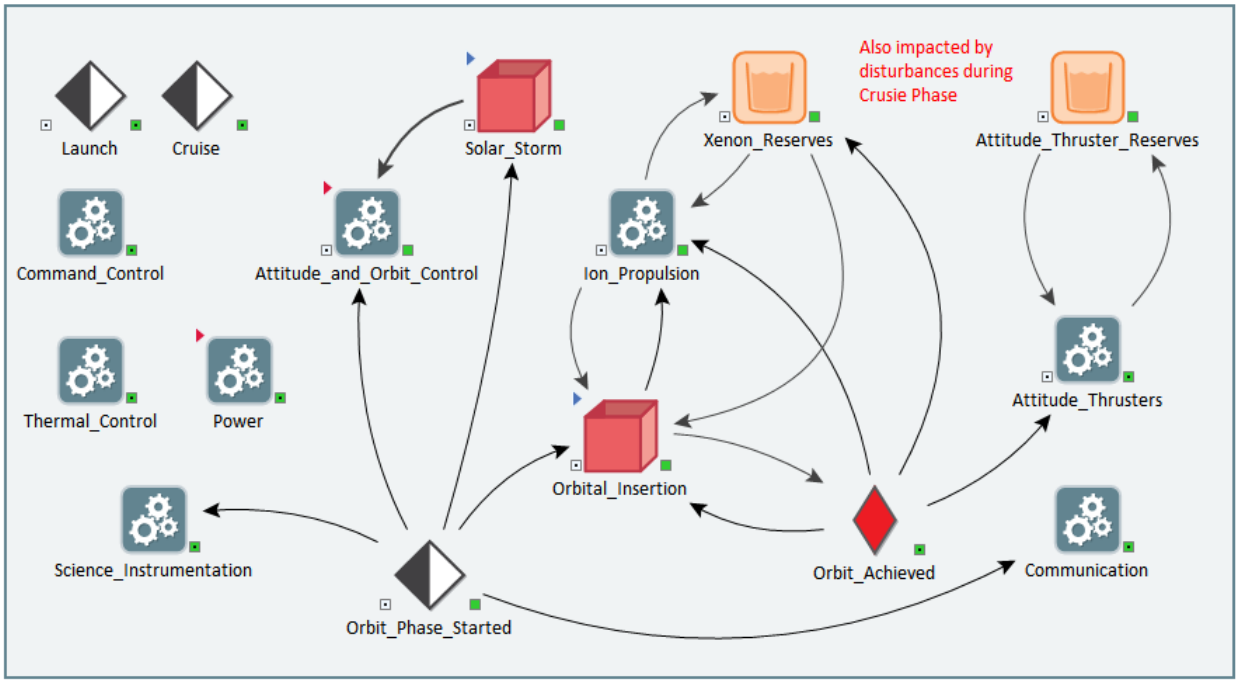
The second PRA model that we will briefly discuss here evaluates the performance of an unmanned scientific mission to another planet. There are a number of phases (launch, cruise, orbital insertion, lander descent and scientific mission on the surface), and the requirements for successful operation change as the mission evolves. Of course, using a simulation approach makes it straightforward to represent these changing requirements. The key outputs from the model are the proportions of missions where minimal and full scientific goals are achieved. Let's briefly examine a couple of these phases.

To successfully launch the spacecraft (consisting of an orbiter and a lander), the rocket engines (solid rocket boosters, main engines and upper stage), must function when required during the launch sequence. They must also successfully separate at the appropriate time. Any failures lead to loss of vehicle. For example, here is the Main Engine submodel within the Launch Phase submodel:



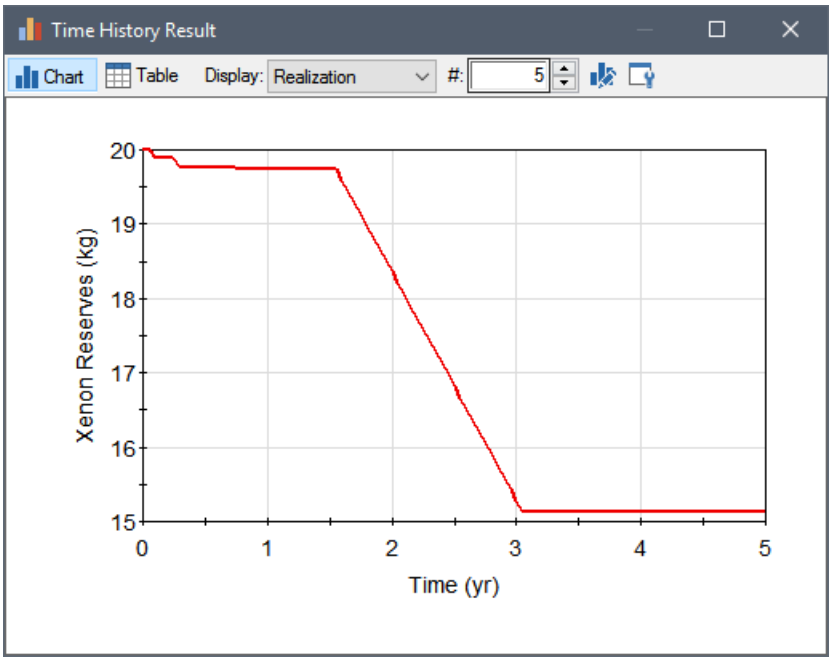
A Reliability element is used to model the main engines, and a Random Choice is used to model the separation.

The orbiter model is shown below:



Note that in addition to containing a number of Reliability elements for the various system components, it also tracks consumables (Xenon_Reserves and Attitude_Thruster_Reserves). A simulation-based approach allows us to directly model consumables (i.e., fuel), and in doing so realistically represent additional mission failure scenarios (e.g., running out of fuel).

During the cruise phase, the path of the spacecraft can be disturbed due to a number of causes (e.g. micrometeoroid debris, solar winds). These "nuances" will require the spacecraft to perform a correction, with its attitude thruster and ion engine consuming a certain amount of their propellant reserves. If the cruise phase is successful, the spacecraft wakes up its remaining systems and begins insertion into orbit. Propellant may also be consumed during orbital insertion. Both the ion engine and attitude thrusters are subsequently required while the orbiter is at the target planet. Therefore, the frequency of nuances during the cruise phase, along with the potential need to use the ion engine to enter orbit, can result in termination of the mission. A typical time history plot of the amount of ion engine (Xenon) propellant is shown below:



In this particular realization, orbit was achieved around 1.5 years (and previous to that, some fuel was used due to cruise phase disturbances), a small amount of fuel was used at insertion and then a constant mount of fuel was used while in orbit. Shortly after orbit is achieved, the lander reached the surface and the scientific mission began. However, just after three years, the orbiter failed (and hence stopped using fuel). As a result the mission ended at that time.

Once on the surface, the scientific mission is intended to last three years. The lander has two key pieces of scientific equipment on board. Successful completion of minimal mission requirements requires that either these two instruments be active for the first year on the surface. Completion of all mission goals requires both instruments to be active for three years on the surface. Hence, due to the failure of the orbiter about 18 months after the beginning of the scientific mission on the surface, the particular realization shown above would have been judged to have met only minimum mission requirements.

The key results for a simulation like this consist of the probability of meeting minimum and full mission requirements. By using GoldSim's causal analysis features, you could then look at those components contributing to failures and potentially modify the design to increase the probability of mission success.

Summary

This document briefly discussed how ***GoldSim***, a dynamic probabilistic simulation program, can be used to tackle complex reliability and risk assessment problems that cannot be easily or realistically addressed using traditional modeling approaches.

In addition to computing traditional metrics for reliability (e.g., reliability and availability) and risk assessment (e.g., the probability of specific consequences), GoldSim also catalogs and analyzes failure scenarios, which allows for key sources of unreliability and risk to be identified (i.e., root cause analysis).

However, the true power of GoldSim is that it can do more than compute only these kinds of reliability and risk management metrics. This is because GoldSim differs from the few existing simulation-based approaches to reliability and risk assessment in that it combines powerful features for representing the failure (and repair) of complex systems with the flexibility to represent the true dynamic complexity and evolution of the entire system. That is, GoldSim is first and foremost a powerful and extremely flexible general-purpose, probabilistic, dynamic simulator that has been used to simulate the behavior and evolution of a wide variety of complex systems ranging from environmental systems (e.g., mines, watersheds, waste disposal sites) to engineered systems (e.g., processing facilities, machines, space missions) to business systems (e.g., companies, projects).

By combining these fundamental capabilities with the Reliability Module, a specialized extension for dynamically modeling the failure (and repair) of engineered components, GoldSim makes it possible to build "total system models" that can represent 1) ***evolving environmental conditions***; 2) the realistic, ***dynamic complexity of failure*** of components within the system (e.g., complex interdependencies, failure rates that respond to evolving environmental conditions); and 3) the actual ***consequences*** of failure (e.g., changes in throughput, costs, loss of life, and other measures of system performance).

An excellent way to explore GoldSim further is to request a free, fully-functional evaluation version of the software from the GoldSim website (www.goldsim.com). When you install the software, PDF documents of the user's guides are installed with the software. This includes the Reliability Module User's Guide, which includes many examples. More detailed example applications (including the lunar base and planetary exploration mission discussed above) are also available for download from the GoldSim Model Library (http://www.goldsim.com/Library/Models/). Finally, you can always send questions to the GoldSim Help Desk at support@goldsim.com.

References

Ebeling, C.E., 2009, An Introduction to Reliability and Maintainability Engineering, Second Edition, Waveland Press, Long Grove, Illinois.

Mattenberger, C.J. et al., 2015, Comparative Analysis of Static & Dynamic Probabilistic Risk Assessment, *2015 Annual Reliability and Maintainability Symposium (RAMS)*, Palm Harbor, Florida.

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, Version 1.1, NASA Headquarters, Washington, D.C.