Simulator User Guide

Vitech

Copyright © 2009-2017 Vitech Corporation. All rights reserved.

No part of this document may be reproduced in any form, including, but not limited to, photocopying, translating into another language, or storage in a data retrieval system, without prior written consent of Vitech Corporation.

Restricted Rights Legend

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.277-7013.

Vitech Corporation
2270 Kraft Drive, Suite 1600
Blacksburg, Virginia 24060
540.951.3322 FAX: 540.951.8222
Customer Support: support@vitechcorp.com
www.vitechcorp.com

GENESYS™ is a trademark of Vitech Corporation and refers to all products in the GENESYS software product family.

Other product names mentioned herein are used for identification purposes only, and may be trademarks of their respective companies.

Publication Date: August 2017

TABLE OF CONTENTS

Preface vii
1 GENESYS simulator – Dynamic Verification Simulator 1
1.1 Modeling and Simulation with GENESYS 2
1.2 Open Simulator 4
1.3 Simulator Window 5
1.4 Transcript Pane 9
1.5 Execution Hierarchy Pane 10
1.6 Database Entities Pane 10
1.7 Simulation Entities Pane 11
1.8 Simulator Preferences 12
2 Introduction to Behavior 15
2.1 Function Flow Block Diagrams (FFBDs) 15
2.2 Enhanced FFBDs 16
2.3 Behavior Elements 17
2.4 NumberSpec 21
2.5 ScriptSpec 23
3 Control Constructs 25
3.1 Parallel 25
3.2 Select 25
3.3 Multi-Exit Function 26
3.4 Exit Node 26
3.5 Loop 27
3.6 Loop Exit 27
3.7 Iterate 27
3.8 Replicate 28
4 Using the GENESYS simulator 29
4.1 Basic Constructs 29
4.2 Items in Models 33
4.3 Iterations 35
4.4 Multi-Exit Functions 37
4.5 Selection Probabilities 38
4.6 Multiple Levels 40
4.7 Loop 43
4.8 Loop Exit 44
4.9 Replicate 45
4.10 Kill Branch 46
4.11 Executing a Model with Links and Items 48
4.12 Execution Order 52
4.13 Resource execution order 52
5 Adding Resources 55
5.1 Executing a Model with Resources 55
6 Enhancing the Simulator Model Using Scripting 59
6.1 Script Basics 59
6.2 GENESYS Application Programmer's Interface 60
Appendix A - GENESYS simulator Distributions 61
Appendix B - Transcript Window Content Description 69

LIST OF TABLES

Table 1 Timeline Window Color Codes 7
Table 2 Transcript Window Contents 10
Table 3 Function Element Attributes 17
Table 4 Item Attributes 19
Table 5 Link Attributes 19
Table 6 DomainSet Attributes 20
Table 7 Exit Attributes 20
Table 8 Resource Attributes 21
Table 9 NumberSpec Attributes 22
Table 10 ScriptSpec Attributes 24
Table 11 Random Variable Definitions 62

LIST OF FIGURES

Figure 1 Example of an Enhanced Functional Flow Block Diagram 2
Figure 2 Simulation Timeline 3
Figure 3 Opening the Simulator Display Window from the Toolbar 4
Figure 4 Opening the Simulator Display Window from the Activity Diagram 4
Figure 5 The Simulator Display Window 5
Figure 6 Simulator Window and Timeline Pane 6
Figure 7 Timeline Controls On Simulator Ribbon 8
Figure 8 Transcript Pane in the Simulator Window 9
Figure 9 Database Entities Pane 11
Figure 10 User Simulator Preferences 13
Figure 11 Project Simulation Preferences 13
Figure 12 User Random Distribution Preferences 14
Figure 13 User Random Stream Preferences 14
Figure 14 A Function Flow Block Diagram 15
Figure 15 An Enhanced Function Flow Block Diagram 16
Figure 16 NumberSpec Type Selector Window 22
Figure 17 Script Editor Window 23
Figure 18 Single Function 29
Figure 19 Single Function Simulation Results 29
Figure 20 Sequence of Three Functions 30
Figure 21 Multiple Functions Simulation Results 30
Figure 22 Select Construct 31
Figure 23 Select Construct Simulation Results 31
Figure 24 Parallel Construct 32
Figure 25 Parallel Construct Simulation Results 32
Figure 26 A Data Store Item 33
Figure 27 Data Store Item Simulation Results 34
Figure 28 A Data Trigger 34
Figure 29 Trigger Item Simulation Results 35
Figure 30 An Iterate Construct 36
Figure 31 Iterate Construct Simulation Results 36
Figure 32 A Multi-Exit Function Construct 37
Figure 33 Multi-Exit Function Construct Simulation Results 38
Figure 34 Multi-Exit Function Construct with Selection Probabilities 39
Figure 35 Multi-Exit Function with Selection Probabilities Simulation Results 40
Figure 36 Second Function Decomposed 40
Figure 37 Model Containing Decomposition 41

Figure 38 Results of Model Containing Decomposition 42
Figure 39 Execute Decomposition Set to False 43
Figure 40 Loop Construct 43
Figure 41 EFFBD with Loop and Loop Exit 44
Figure 42 Results for Model with Loop and Loop Exit 45
Figure 43 A Replicate Construct 45
Figure 44 Results for Model with A Replicate Construct 46
Figure 45 Model with Kill Branch 47
Figure 46 Simulation Results with Kill Branch 48
Figure 47 Interface and Link Schema Diagram 49
Figure 48 Link-Item Model 50
Figure 49 Simulation Results with Item Size = 0.0, Link Capacity = 1000.0, Delay = 0.0 50
Figure 50 Simulation Results with Item Size = 1000.0, Link Capacity = 1000.0, Delay = 0.0 51
Figure 51 Simulation Results with Item Size = 1000.0, Link Capacity = 1000.0, Delay = 0.5 51
Figure 52 Loop Model with Insufficient Resources 56
Figure 53 Loop Model with Adequate Resources 56
Figure 54 Loop Exit Model with Trigger-Type Resource 57
Figure 55 Edit ScriptSpec Window 59

CUSTOMER RESOURCE OPTIONS

Supporting users throughout their entire journey of learning model-based systems engineering (MBSE) is central to Vitech's mission. For users looking for additional resources outside of this document, please refer to the links below. Alternatively, all links may be found at www.vitechcorp.com/resources.

Webinars

Webinar archive with over 40 hours of premium industry and tool-specific content.

Screencasts

Short videos to guide users through installation and usage of Vitech software.

A Primer for Model-Based Systems Engineering

Our free eBook and our most popular resource for new and experienced practitioners alike.

Help Files

Searchable online access to Vitech software help files.

Technical Papers

Library of technical and white papers for download, authored by Vitech systems engineers.

MySupport

Knowledge Base, Exclusive Webinars and Screencasts, Chat Support, Documents, Download Archive, etc.

Our team has also created resources libraries customized for your experience level:

All Resources	**Advanced**
Beginner	**IT / Sys Admin**
Intermediate	**Student**

PREFACE

The purpose of this guide is to provide the user with an understanding of the features and behavior of the GENESYS™ simulator, the discrete event simulator that is part of the GENESYS product suite.

The views in the guide reflect the latest version of GENESYS.

This guide assumes the user has attended the "Model-Based Systems Engineering with GENESYS" course offered by Vitech and has been introduced to the basics of GENESYS and the simulator. This document is intended to reinforce this basic information and provide more detailed instruction on the GENESYS simulator.

This document is organized into six sections.

- Section 1 provides an introduction to the GENESYS simulator followed by a discussion of the various commands the user will need in order to use the simulator.
- In Section 2 modeling of behavior is discussed; this section discusses the GENESYS entity classes and their associated attributes that are necessary to model this behavior.
- Section 3 provides the user with a refresher on the various control constructs used in the simulator.
- Section 4 provides behavior response examples in the form of simulator timelines. Starting with simple examples, complexity is added to demonstrate the behavior of all of the constructs described in Section 3.
- Section 5 focuses on the use of resources in behavior model.
- Lastly, Section 6 provides a brief introduction to the use of Microsoft Visual Basic for scripting within the GENESYS simulator.

The following conventions are used throughout this guide to aid the user:

Example	Description
Function	Entities
Duration	Attributes
consumed by	Relationships
true	User Selections and Input
File > Import	Reference Documents, GENESYS Commands, Buttons, Icons, or Tabs

THIS PAGE INTENTIONALLY BLANK

1 GENESYS SIMULATOR – DYNAMIC VERIFICATION SIMULATOR

To completely define a system, one must precisely specify the behavior model detailing the functional and control flows, the inputs and outputs, the physical architecture model, and the performance model.

The inputs to this process include requirements, and the outputs include a system design model. The system design model must satisfy all the requirements in a logical manner so that the system is dynamically consistent. If the behavioral model is flawed and not corrected during the initial design stages, problems will likely multiply during development. Inevitably, when achievement of dynamic consistency is left until the end of a project (during integration and testing), it is the least attainable and the most costly to correct. An automatically generated dynamic verification simulator can help identify flaws early in the development cycle.

The GENESYS simulator is a simulation engine that provides this automatic dynamic verification of operational and/or functional behavior models constructed as part of a system, architecture, program, or process definition in GENESYS. The GENESYS simulator provides the ability to evaluate designs from two perspectives:

1. **Correct logical behavior.** Verify that the design as modeled will in fact execute logically and that activities and functions will occur in the sequence intended. This is of particular importance as an architecture or system design grows more complicated with multiple layers of decomposition and many parallel tasks. As designs grow more complex, it becomes difficult or impossible to validate logical behavior through visual inspection alone. Execution of behavior in the GENESYS simulator can and should be a significant contributor to the task of verification of your logical designs.

2. **System/architecture performance**. Real-world limitations on (and competition for) resources, architecture design selection and functional allocation can all act to constrain the overall performance of a chosen solution. These constraints can be captured in the GENESYS model and their aggregate effects on performance evaluated by executing behavior models in the GENESYS simulator. Process queuing, resource utilization, and communications bandwidth are some of the examples of what can be seen across a timeline, with the results utilized to correct and/or optimize architecture and system designs.

A fundamental benefit of using the GENESYS simulator is the automatic generation of a simulation from the system operational or functional model ensuring that the simulation model and system functional model remain consistent. If changes are made to the simulator, those changes are automatically reflected in the system model. This differs from classic stand-alone simulators in which the simulation itself is tuned with changes often not reflected in the system specification. The net result is virtual prototyping at the system level directly from the system model.

1.1 Modeling and Simulation with GENESYS

This guide provides information on the GENESYS simulator, an option available to enhance the GENESYS systems engineering and architecting tool. The guide assumes some understanding of the concepts of systems engineering, architecting, and functional modeling as well as a basic familiarity with GENESYS commands. Additional background on these concepts can be found in the ***GENESYS System Definition Guide***, ***GENESYS Architecture Definition Guide***, and ***GENESYS Guided Tour***.

The GENESYS simulator provides the capability to simulate behavior – whether represented by activity diagram, Enhanced Functional Flow Block Diagram (EFFBD), or any other behavior representation – in order to analyze the dynamic performance and behavior of your system‘s functional model. The term “Functional” in this case is a broad term used to describe the type of diagram. In GENESYS this type of diagram is used to model behavior for system Functions, Operational Activities, Test Activities, and Program Activities, entities of the GENESYS classes **Function, OperationalActivity**, **TestActivity**, and **ProgramActivity**, respectively. In subsequent examples, “Function” can be assumed to also mean “Operational Activity” (for architecture development), “Test Activity” (applicable for testing management), or “Program Activity” (applicable for program management modeling). With the GENESYS simulator, you can directly assess resource availability, verify the functional logic, and assess system performance and resource utilization using the behavior specification of your system.

The Enhanced Functional Flow Block Diagram, shown in Figure 1, provides a basis for establishing a multi-layered process model. These models represent sequential, parallel, repetitive, and decision logic used to depict system behaviors. A simulator may be opened for any functional model – complete or incomplete, interim or final – and run.

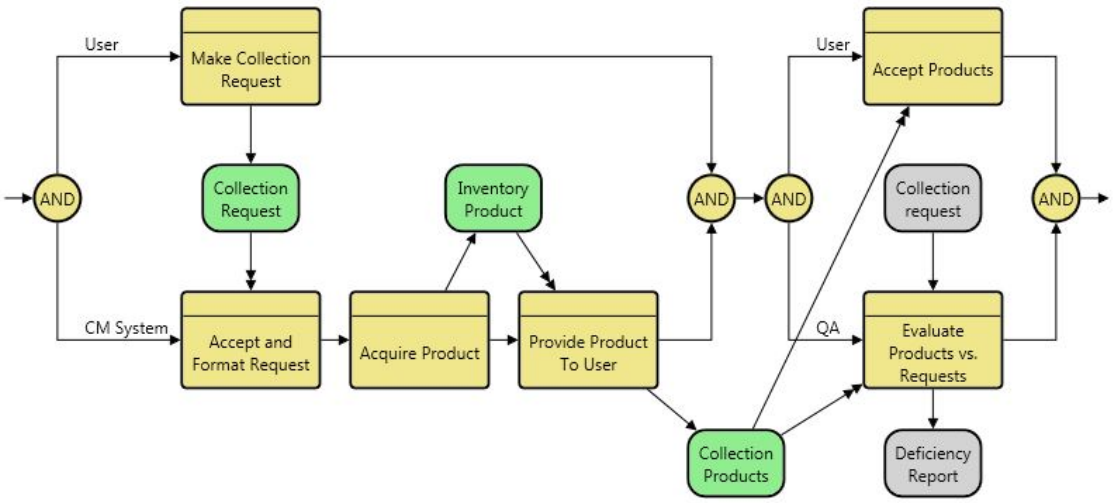


Figure 1 Example of an Enhanced Functional Flow Block Diagram

The simulator identifies the discrete events that occur during the simulation execution and constructs a timeline that depicts:

- Functional activation, sequential, parallel or triggered execution, and duration
- Wait states
- Resource inventory history
- Branching selection (based upon random probabilities, or specified criteria)
- Queuing triggers (input items waiting to be processed by functions)

The GENESYS simulator generates a timeline for execution and identifies when **Functions** are initialized, how long they wait for a trigger (**Item**) or **Resource** to be available, the time it takes to perform a **Function**, and which paths or branches are taken based upon specified probabilities. Figure 2 illustrates the simulation timeline for the EFFBD shown in Figure 1.

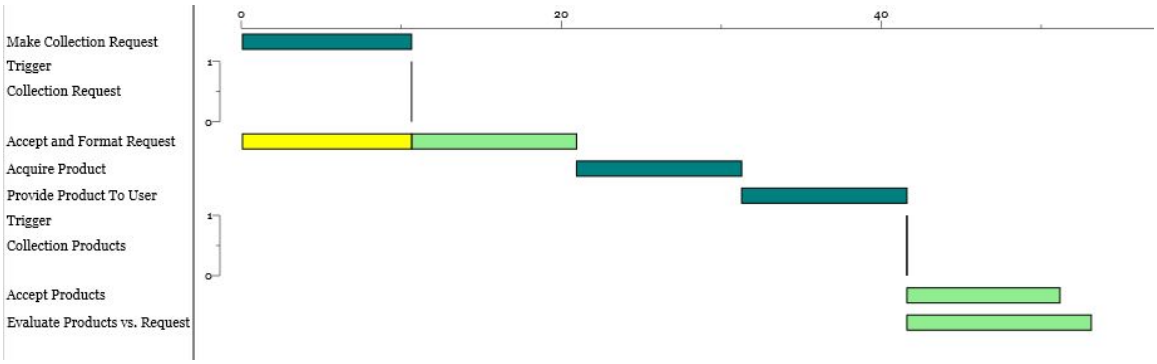


Figure 2 Simulation Timeline

The next sections describe how to run the GENESYS simulator and describe the details of the various simulator windows.

1.2 Open Simulator

The simulator executes the behavior of a function.

A simulator can be opened using either of the following methods:

In the Project Explorer, select the **Function,** or the related Component[1] or OperationalNode, to be dynamically analyzed by the simulator, then from the ***Home*** ribbon click the ***Simulator*** button as shown in Figure 3**.**

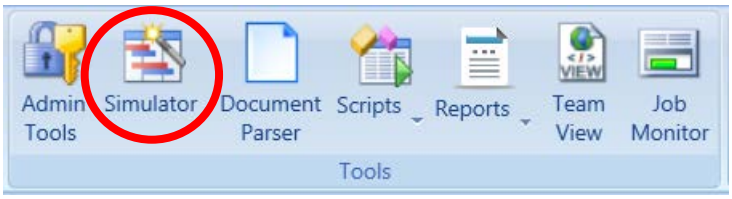


Figure 3 Opening the Simulator Display Window from the Toolbar

or

Click the ***Simulator*** icon on the ***Diagram*** ribbon of an activity diagram, FFBD, EFFBD, or sequence diagram opened on the top-level function of interest as shown in Figure 4.

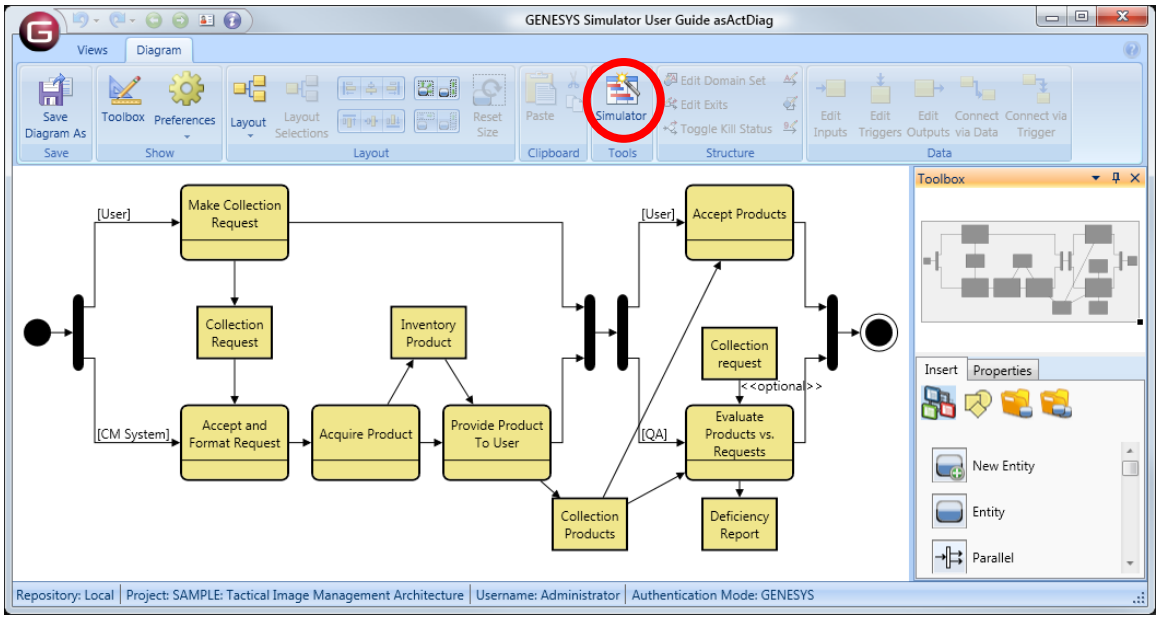


Figure 4 Opening the Simulator Display Window from the Activity Diagram

[1] When a Component or OperationalNode entity has a Function related to it using the Performs relation with the relation attribute Behavior Type set to Integrated (Root), the Simulator will dynamically analyze that Function.

When the simulator is first opened, the simulator window will appear. As shown in Figure 5, the simulator window includes a ribbon with a ***Simulator*** tab and a ***Views*** tab. The simulator window includes the timeline pane and, depending on the user preference settings, the transcript pane. The timeline pane is the primary pane and independent of the transcript pane. If desired, the transcript pane may be closed without closing the timeline pane. However, if the simulator window is closed, all the associated simulator panes are also closed. The features of each of these panes will be discussed in detail in the following subsections.

The simulator window is associated with the **Function** for which the simulator was opened. Thus, more than one simulator window can be open at a time, each window associated with a particular **Function**. The label on the top line of the simulator window identifies this **Function**.

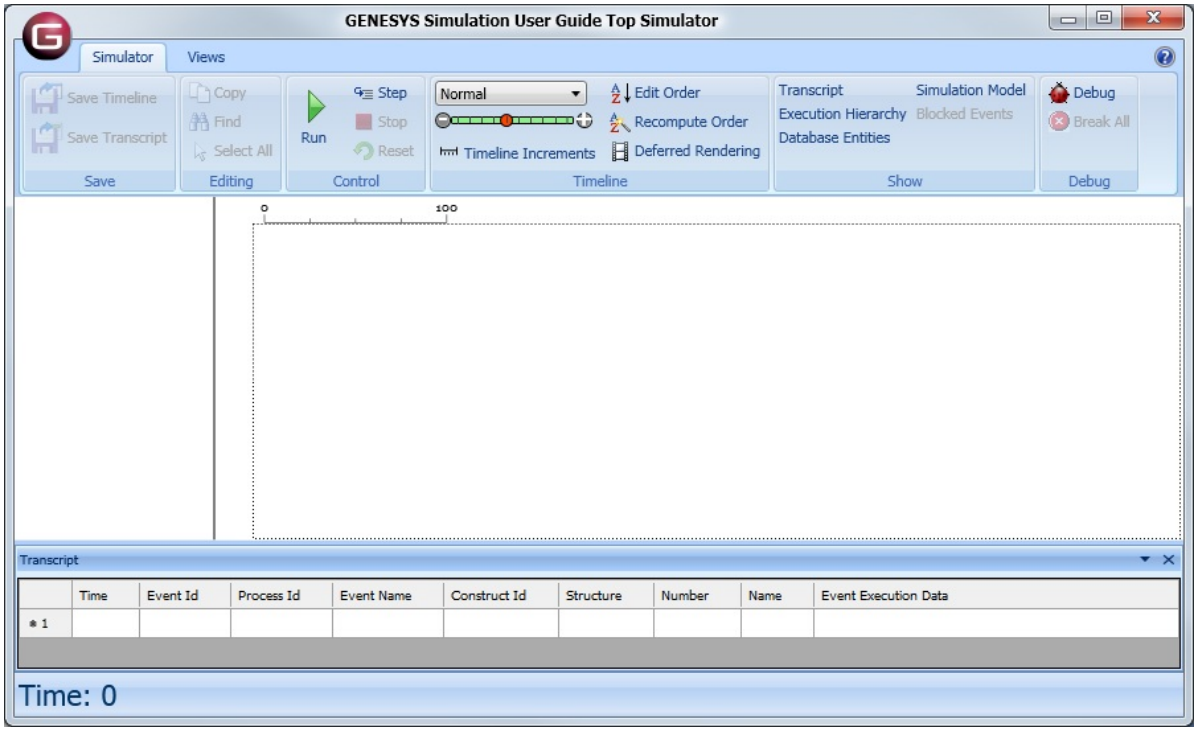


Figure 5 The Simulator Display Window

1.3 Simulator Window

Figure 6 shows the simulator window. The primary area of display within this window is the actual simulation timeline. In addition to the timeline, the simulator window also includes the Simulator ribbon containing the simulator control buttons. The simulator timeline displays what is happening over the course of a simulation run.

Depending on both the entities contained in the model and objects selected by the user, the timeline will display **Functions** as they occur during the execution of a behavior model, levels of **Resources** that are utilized during the simulation run, queuing of triggers (**Items**) waiting to be processed by a **Function**, and the throughput of **Links** in terms of the Capacity attribute.

The timeline pane is subdivided into two panels. On the left side, there is a listing of any **Resources** used in the model, the **Functions** as they are encountered, **Link** capacity, and any **Item** queue, depending on entities selected to appear by the user and in the order edited by the user. On the right side, the events or status are displayed chronologically in the simulation timeline for each of these elements listed. Note, the timeline will not list **Functions** that are not executed because of the deterministic or probabilistic logic that controls branching or exit selection.

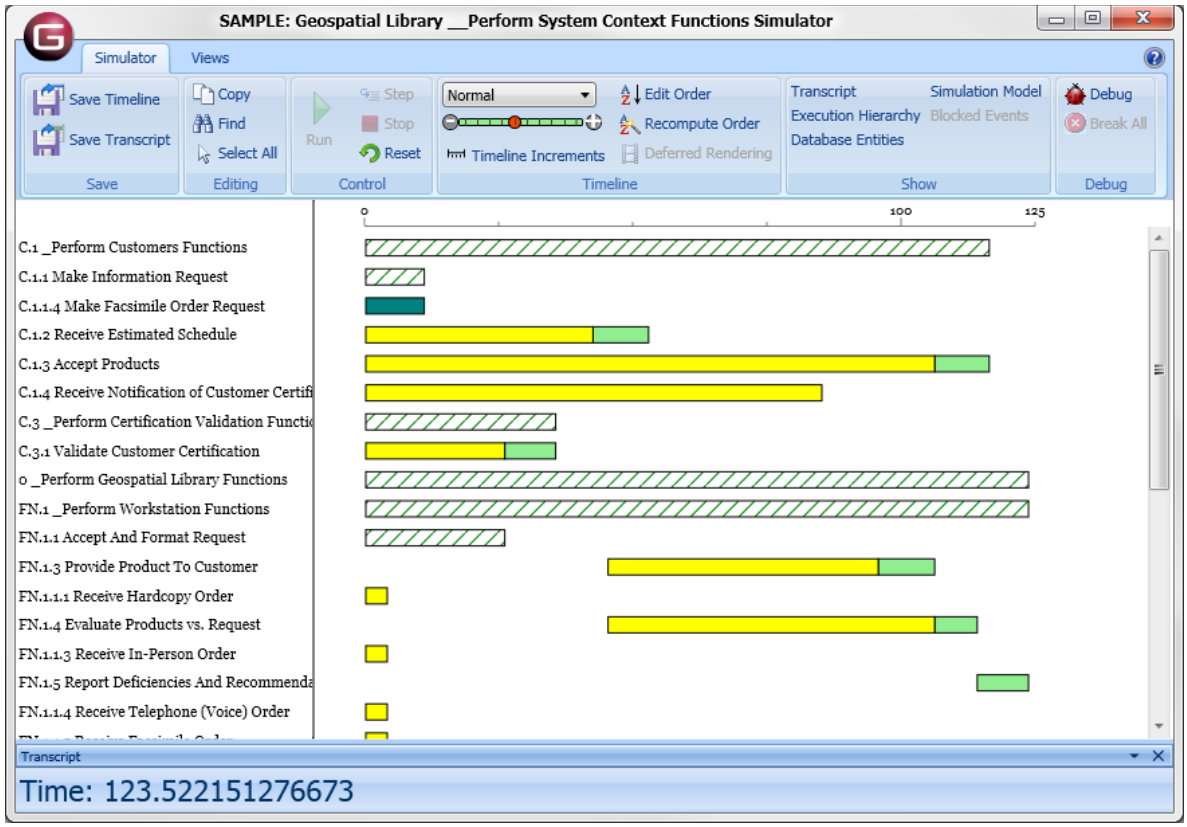


Figure 6 Simulator Window and Timeline Pane

When a simulation is run, colored duration bars will appear in the timeline. The GENESYS simulator uses these colors to distinguish between different types of events. Table 1 describes the color codes. These will make more sense as the types of events are described in a model execution in Sections 3 and 4.

Timeline Output
GENESYS allows simulation timelines to be viewed, printed and/or saved to a file. Timelines can be saved as PNG, JPEG, Bitmap, GIF, TIFF or XPS files.

Table 1 Timeline Window Color Codes

	Teal	The **Function** is executing after obtaining any needed **Resources** and without having to wait for a trigger.
	Bright Green	The **Function** is executing after the trigger(s) arrived.
	Yellow	The **Function** is enabled but it has not yet initiated execution. It is waiting for a trigger.
	Magenta	The **Function** is enabled but waiting for **Resources**.
	Diagonal	The **Function** is decomposed and its behavior is being executed.
	Grey	The amount **of Resource** available. This varies as it is *consumed*, *captured*, or *produced* by a **Function**.
	Red	The amount of **Resource** reserved for a **Function**. The **Resource** Amount Available is not sufficient to enable the function to execute.
	Turquoise	A trigger has arrived and the triggered **Function** is not enabled resulting in queuing of the **item**. The trigger is displayed just above the triggered **Function**.

1.3.1 Simulator Ribbon

The commands in the simulator ribbon control the execution of the simulator. It is from here that the user starts execution, steps through the execution, stops execution, and resets the simulator. The commands are grouped into six sections: Save; Editing; Control; Timeline; Show; and Debug.

In the **Save** section:

Save Timeline allows simulation timelines to be saved to a file. Timelines can be saved as PNG, JPEG, Bitmap, GIF, TIFF or XPS files.

Save Transcript allows the simulator transcript to be saved to a CSV file.

In the **Editing** section:

Copy allows text to be copied from the panes, such as the Transcript pane.

Find allows text in the panes to be searched.

Select All enables all the text in a pane to be selected.

In the **Control** section:

Run the simulation begins or continues execution of the Function's behavior model.

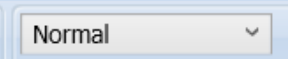
Step through the simulation advances the simulator to the time of the next event to occur as defined by the behavior model and executes all events that occur at that time.

Stop the simulation halts execution, particularly useful when in an undesired loop.

Reset the simulation clears the simulator clock, the timeline window, and the transcript window in preparation for a new run. The simulator must be reset before subsequent executions.

The controls in the Timeline section, of the Simulator ribbon, allow the timeline pane to be configured by the user. Options include changing the relative scale of the timeline, the order of appearance of the elements used in the simulations, the relative sizes of the panes, and the time scale increment:

Figure 7 Timeline Controls On Simulator Ribbon



Timeline Major Scale changes the time scale increment. The width of the timeline displayed in the right-hand pane of the timeline window can be adjusted using the Timeline Major Scale drop-down selection. The timeline is adjusted as a new setting is selected. Normal is the initial default setting, but this default setting can be changed from the User Preferences (see User Simulation Preferences). Different scales are possible depending on the situation. At times, it may be important to view the details, in which case use the Large setting. At other times, it may be desirable to view the complete timeline on the screen, in which case the Screen setting or the Small setting can be used.

Timeline Minor Scale permits selecting the amount of timeline to display via a slide bar.

Edit Order opens a dialog listing the entities in the timeline. Use the up-down arrows to change the order of appearance of an entity in the timeline. The entities generally appear in the timeline list in the order that they are encountered in simulation execution. It may be desirable to reorder the rows in the timeline using the Edit Order dialog box. Re-ordering the list does not alter the order of execution but rather just alters the display.

Recompute Order returns the entities to their natural execution order. When a Function or Resource is added to the list using the Database Entities pane (discussed in Section 1.6), the new entity will appear at the bottom of the list of entities. Use Recompute Order to include the Function in the order in which it was encountered.

Timeline Increments allows the major and minor timeline tick increments to be changed by the user. The default values can be set in the User Preferences (see User Simulation Preferences).

Deferred/Realtime Rendering button allows the user to choose whether the timeline is updated as the simulation executes, in Realtime mode, or whether the timeline is displayed when the simulation has finished executing, in Deferred mode.

In the **Show** section:
Transcript opens the simulation Transcript pane if it is closed.
Execution Hierarchy opens the simulation Execution Hierarchy pane if it is closed.
Database Entities opens the Database Entities pane to allow the user to select which entities are displayed in the simulation timeline.

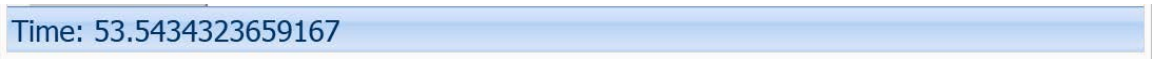
Simulation Entities opens the Simulation Entities pane if it is closed.
Blocked Events opens the simulation Blocked Events pane if it is closed.

In the **Debug** section:

Debug places the simulator in Debug mode.

Break All sets debugging break points in the simulation model.

Result Detail



The ***Result Detail*** area at the bottom of the simulator window provides the total duration, in elapse time units, for the execution of the simulation.

1.4 Transcript Pane

The simulator transcript pane (
Figure 8) displays each event that occurs during the simulation and the time (on the simulator clock) when the event occurred. In general, the transcript records the time each event occurred, the type of event (started, enabled, waiting for resources, etc.), and the number and name of the element involved in the event. The transcript pane provides a basis for understanding the detailed execution of the behavior model. From this pane, you can review the detailed events as they occurred during the execution of the simulator.

Transcript

	Time	Event Id	Process Id	Event Name	Construct Id	Structure	Number	Name	Event Execution Data
01	0	ID(1)	PROC(0)	Start	1	Parallel			
02	0	(aux)	PROC(1.1)	Enabled	1.1.1			Make Collection Request	Enabled: Make Collection Request
03	0	(aux)	PROC(1.1)	WaitingForResources	1.1.1			Make Collection Request	Waiting for Resource Make Collection Request
04	0	ID(2)	PROC(1.1)	Start	1.1.1			Make Collection Request	
05	0	(aux)	PROC(1.2)	Enabled	1.2.1			Accept and Format Request	Enabled: Accept and Format Request
06	10.5690245683168	(aux)	PROC(1.1)	Write	1.1.1			Make Collection Request	Write: Collection Request
07	10.5690245683168	(aux)	PROC(1.1)	Queued	1.1.1			Make Collection Request	Queued: 1.1.1,Make Collection Request,Collection R...
08	10.5690245683168	ID(5)	PROC(1.1)	Finish	1.1.1			Make Collection Request	
09	10.5690245683168	(aux)	PROC(1.2)	WaitingForResources	1.2.1			Accept and Format Request	Waiting for Resource Accept and Format Request
10	10.5690245683168	(aux)	PROC(1.2)	Dequeued	1.2.1			Accept and Format Request	Dequeued: 1.2.1, Accept and Format Request, Colle...
11	10.5690245683168	ID(3)	PROC(1.2)	Start	1.2.1			Accept and Format Request	
12	10.5690245683168	(aux)	PROC(1.1)	Write	1.1.1			Make Collection Request	Write: Collection Request
13	10.5690245683168	(aux)	PROC(1.1)	Queued	1.1.1			Make Collection Request	Queued: 1.1.1,Make Collection Request,Collection R...
14	10.5690245683168	ID(5)	PROC(1.1)	Finish	1.1.1			Make Collection Request	
15	10.5690245683168	(aux)	PROC(1.2)	WaitingForResources	1.2.1			Accept and Format Request	Waiting for Resource Accept and Format Request
16	10.5690245683168	(aux)	PROC(1.2)	Dequeued	1.2.1			Accept and Format Request	Dequeued: 1.2.1, Accept and Format Request, Colle...
17	10.5690245683168	ID(3)	PROC(1.2)	Start	1.2.1			Accept and Format Request	
18	20.8808039642318	ID(6)	PROC(1.2)	Finish	1.2.1			Accept and Format Request	
19	20.8808039642318	(aux)	PROC(1.2)	Enabled	1.2.2			Acquire Product	Enabled: Acquire Product

Time: 53.0514879241213

Figure 8 Transcript Pane in the Simulator Window

There are nine columns of data contained in the transcript pane. A brief description of each of these columns is presented in Table 2; a more detailed explanation of the contents is contained in Appendix B.

Table 2 Transcript Window Contents

Column	Title	Contents
1	Time	A floating-point number showing the simulation time at which the event occurred.
2	Event ID	Before an event can be executed at a scheduled time, it must first be added to the schedule. As each primary event is created and added to the schedule, it is assigned a sequential event ID, which is shown in Column 2.
3	Process ID	A process represents the execution of a branch in the model. Processes are arranged in a parent-child hierarchy. Each process is assigned a sequential hierarchical ID number based on the ID of the parent.
4	Event Name	This column lists the event type. Many events are related to each other and occur in pairs or higher-order groupings.
5	Construct ID	When the GENESYS simulator builds a simulation model from a GENESYS behavior model, each construct and branch in the simulation model is assigned a unique ID. The fifth column lists the ID of the construct responsible for each event.
6	Structure	If the event is executing a control construct then the name of the structure being processed is displayed in this column. If the event is processing an entity then this column is blank.
7	Number	If the event is executing an entity then the number of the entity being processed is displayed in this column. If the event is processing a structure then this column is blank.
8	Name	If the event is executing an entity then the name of the entity being processed is displayed in this column. If the event is processing a structure then this column is blank.
9	Event Execution Data	The format of ninth column depends on the Event Name and the type of construct that generated the event. The possible execution data formats specific to each kind of event are listed in Appendix B.

1.5 Execution Hierarchy Pane

The Execution Hierarchy pane contains the top level Function's structure (and descendant structures) shown as an indented hierarchy. You can expand or collapse a particular node by clicking the symbol to the left of the node. A red flag shows branches, constructs, and Functions that have not been executed. The flag turns green once the node is executed.

1.6 Database Entities Pane

The **Database Entities** pane (Figure 9) allows you to select the entities to be displayed in the timeline. For instance, you can choose whether or not to show all of the **Resources** in the model.

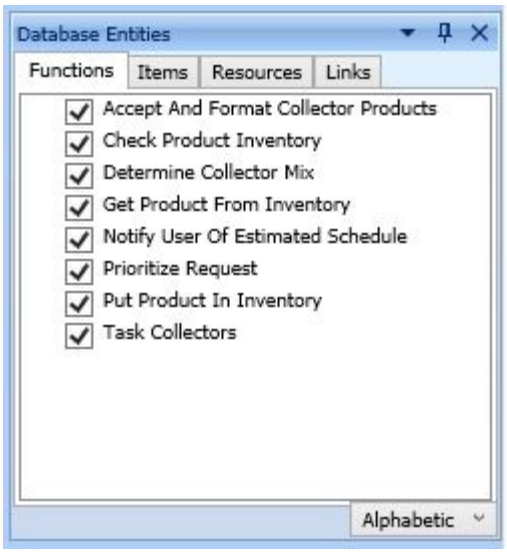


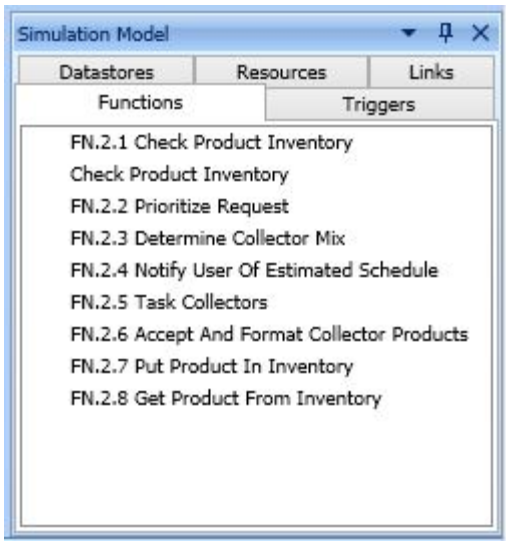
Figure 9 Database Entities Pane

The **Database Entities** pane lists all **Functions**, **Items**, **Links**, and **Resources** included in the model. You may select which entities you want displayed in the timeline by checking or unchecking the entity. If an entity is not checked, that element will not be displayed in the timeline. By default, the ***User Preferences*** settings indicate which classes of entities are automatically displayed.

The lists of entities on each of the ***Functions***, ***Items***, ***Links***, and ***Resources*** tabs can be sorted according to the criteria set using the drop-down menu in the lower right-hand corner of the pane.

1.7 Simulation Entities Pane

The ***Simulation Entities*** pane lists all **Functions**, Triggers (**Items**), Datastores (Items), **Resources**, and **Links** in the model. This is different from the ***Database Entities*** pane in that you can choose whether or not to display each occurring instance of an entity. This is particularly helpful when you want to focus on a particular instance of an entity but do not care about the other instances at this time. A [T] is used to identify the entities being Tracked and hence displayed in the timeline window. Right-click on the entity in order to Set Track, Release Track, or Toggle Track for the selected entity. This is useful when attempting to view the timeline that involves a large number of entities. By turning off the tracking of selected entities, the timeline can be customized to display only those entities of interest.



1.8 Simulator Preferences

There are some preferences that can be preset when working with the simulator. The preferences are accessed from ***GENESYS > Preferences***. These preferences do not change the current values, but when a new simulator window is opened or a new entity is created, the preference settings can be used. In this section the preferences related to the simulation will be addressed, as well as ***Random Number*** items.

The ***User Simulation*** preferences (
Figure 10) allow users to select the GENESYS simulator panes the user would like to automatically see when opening a simulator. [Note the timeline pane will always be open.]

The ***User Simulation*** preferences also allows classes of entities to be set that will initially be displayed in the timeline pane. Users can also choose to display intermediate **Functions** (those **Functions** whose decompositions are executed), set the initial scale, and set the initial tick increments. These default settings will become the initial settings each time the simulator window is opened. However, all these values can also be changed independently in the simulator timeline window once it is opened.

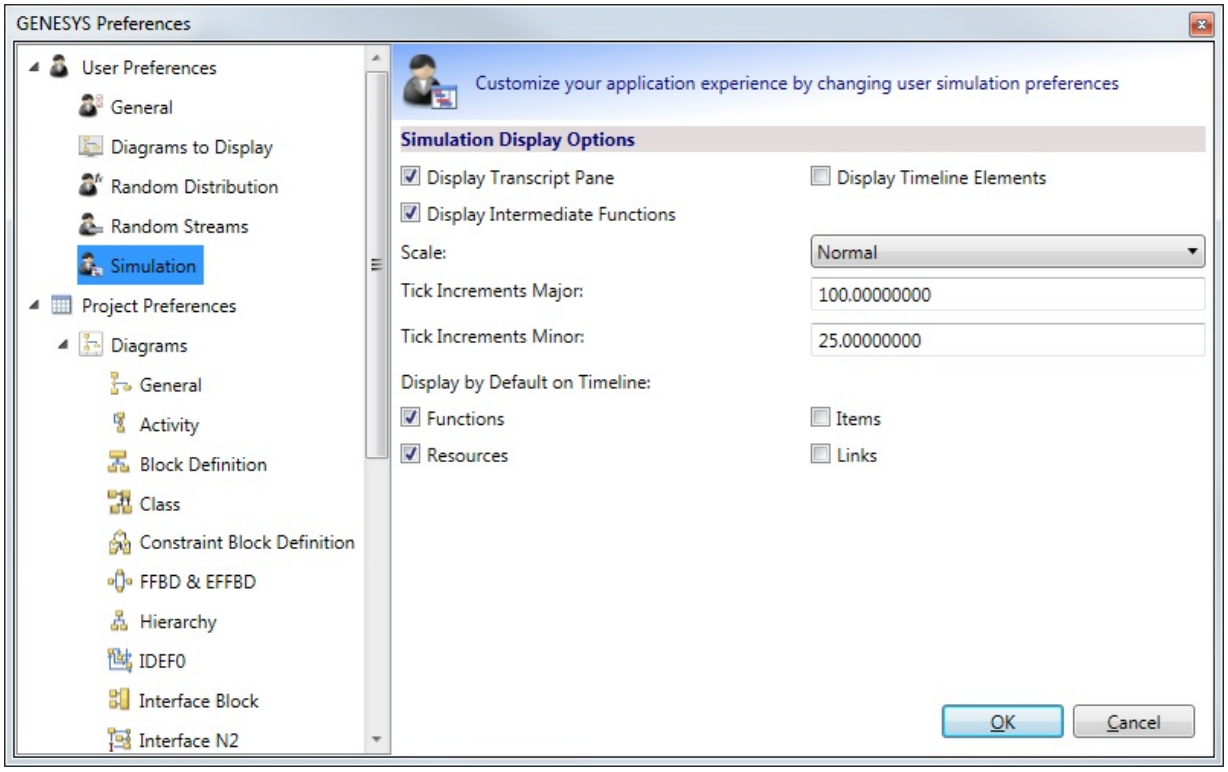


Figure 10 User Simulator Preferences

The ***Project Simulation*** preferences (Figure 11) allows ***Default Iterations*** to be set (the default number of times an Iterate construct will loop) and the ***Default Duration*** (the default time units a **Function** executes). For now, keep in mind default values can be set.

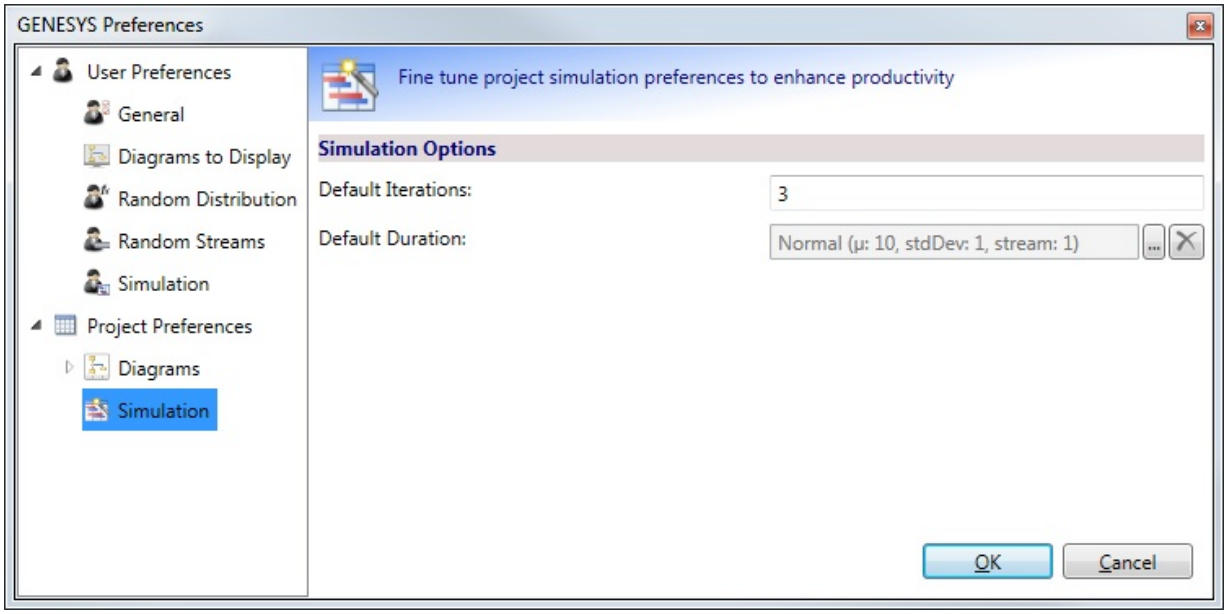


Figure 11 Project Simulation Preferences

The ***User Random Distribution*** preferences (Figure 12) allows default values to be set when a random

distribution value is selected for the **Function** Duration. There will be more on this when the Duration attribute and the NumberSpecs are discussed in Section 2. Keep in mind, every newly created **Function** will have these default values when ***Random*** is selected from the Edit NumberSpec dialog window.

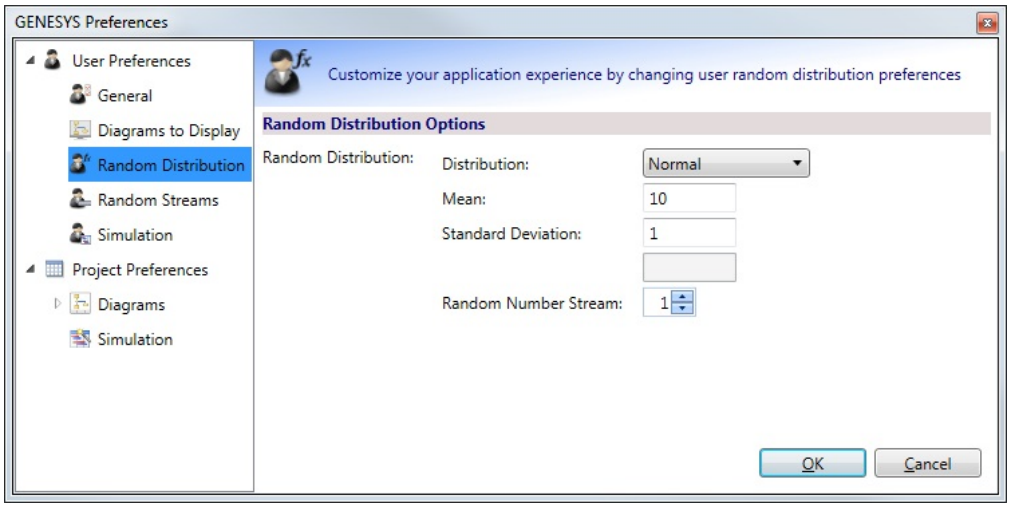


Figure 12 User Random Distribution Preferences

The ***User Random Streams*** preferences (Figure 13) allow repeatability to be assigned to an object. To establish a repeatable simulation run, users must 1) assign a stream to each random duration or resource amount that is of concern, and 2) establish and save a profile. The profile saves the seed upon which the random number generator operates for each assigned stream and will cycle through each stream assigned each time the simulator is executed. Use the up and down arrows to scroll through the 100 streams. Stream [100] Branch is saved for branch selection logic.

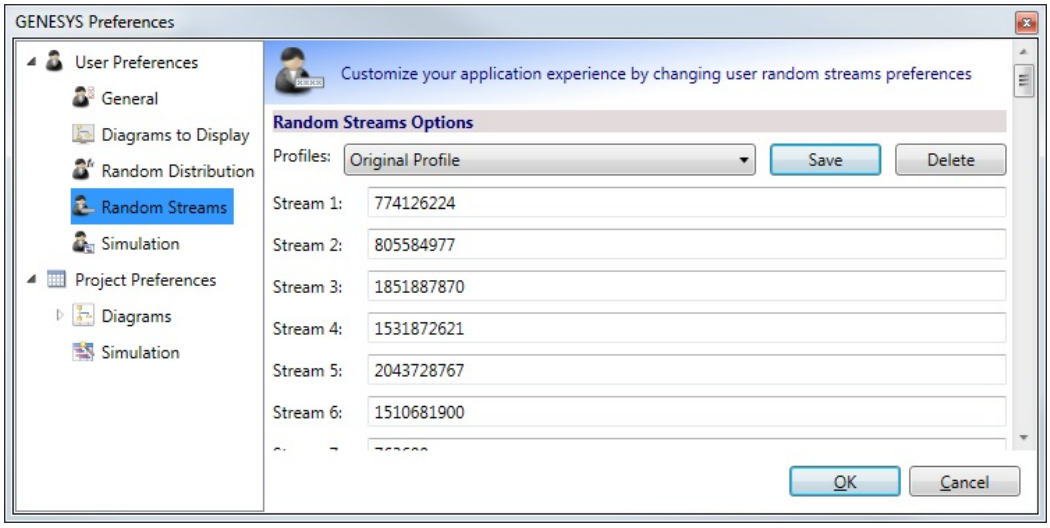


Figure 13 User Random Stream Preferences

2 INTRODUCTION TO BEHAVIOR

To completely define a system, the behavior model must be precisely specified detailing the functional and control flows, the inputs and outputs, the physical architecture model, and the performance model.

GENESYS supports behavior analysis of a system or component of the system. The Function Flow Block Diagrams (FFBDs) and Enhanced Function Flow Block Diagrams (EFFBDs) support specification of required functional sequencing. The FFBD displays the control dimension of the behavior model, whereas the EFFBD includes the data dimension of the integrated behavior model.

2.1 Function Flow Block Diagrams (FFBDs)

The Function Flow Block Diagrams (Figure 14) supported by GENESYS have the classic features of logic structures and hierarchical representations. The constructs allow users to indicate the control structure and sequencing relationships of all functions accomplished by the system being analyzed, including concurrency, branching, iteration, and looping. With Boolean logic specified at decision points, these FFBDs, in combination with the functions' inputs and outputs, form an executable behavior model, allowing dynamic validation via discrete event simulation methods.

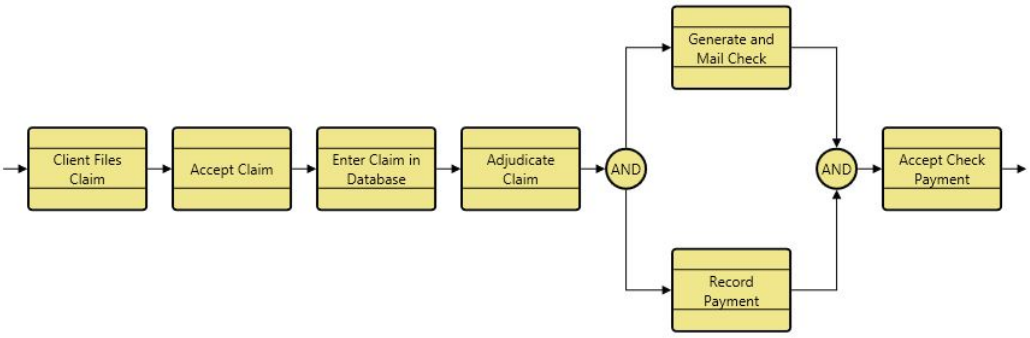


Figure 14 A Function Flow Block Diagram

GENESYS also supports the standard hierarchical features of FFBDs, including the decomposition of a function displayed as an FFBD at the next lower level.

Black Box Notation
When the decomposition of a Function has been defined, the icon representing that Function has a black box in the upper-left corner as a visual cue. In other words, the Function has sub-functions.

2.2 Enhanced FFBDs

The Enhanced Function Flow Block Diagram represents the behavior for a system or component of a system, as shown in
Figure **15**. The EFFBD displays the control dimension of the behavior model in an FFBD format with a data flow overlay to effectively capture behavior in a single, synchronized view.

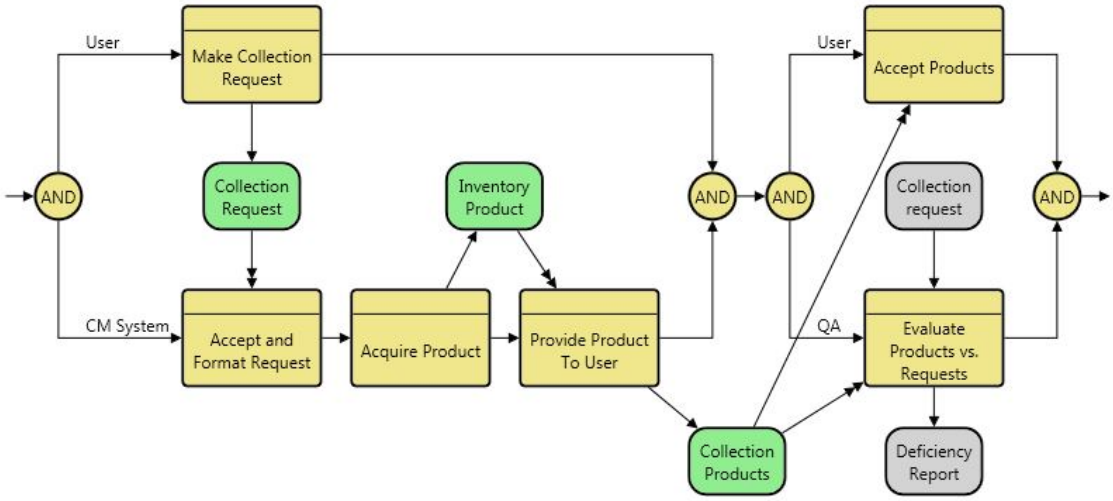


Figure 15 An Enhanced Function Flow Block Diagram

2.2.1 Triggers vs. Data Stores

When displaying the data flow as an overlay on the control flow, GENESYS distinguishes between the two primary types of **Items**, triggers, and data stores. **Items** that control the execution of a **Function** by their presence or absence are designated as triggers. A **Function** begins execution when it has received all of its triggers and its necessary **Resources** have been acquired. Therefore, triggers are **Items** with control implications. **Items** that are not triggers are designated as data stores. Data stores represent inputs to and outputs from **Functions** that are non-triggering and have no effect on control.

In order to clearly represent the behavior of triggers and data stores, the following rules are applied when displaying Items in GENESYS:

- Trigger Items
  - Triggers are shown with green backgrounds by default.
  - Triggers are globally connected with double-headed arrows at the sinks.
- Data Store Items
  - Data stores have gray backgrounds by default.
  - Data stores are globally connected with single-headed arrows at the sinks.

2.3 Behavior Elements

The GENESYS classes that pertain to the behavior of the system are **Function**, **Item**, **Link, DomainSet**, **Exit**, and **Resource**. The equivalent GENESYS classes of **Function**, **Item** and **Link** that pertain to the operational behavior of an architecture using the Department of Defense Architecture Framework (DoDAF) schema are **OperationalActivity**, **OperationalInformation**, and **Needline.** In the discussions below, the information provided applies to both the functional and operational classes, and can be used by both the Systems Engineer and Architect in developing and analyzing the functional and operational behaviors of the system and architecture. For these classes, certain entity and relationship attributes determine how the system performs, while other attributes are informational and pertinent to the system documentation. When running GENESYS simulator, the focus is on the behavior of the system, so only the possible settings for those attributes that affect simulation execution will be discussed in this guide.

2.3.1 Function

A **Function** is a transformation that accepts one or more inputs/triggers (**Items**) and produces outputs (**Items**). It is decomposable, and its behavior is defined using an EFFBD. When executed by the simulator, certain attributes of a **Function** (and attributes of relationships between a **Function** and **Resources** and **Exits**) affect its execution. These are described in Table 3 below. An order has been established for determining when these attribute values are executed for a given **Function** in a GENESYS simulator model. This order is discussed in Section 4.12.

Table 3 Function Element Attributes

ATTRIBUTE	DATA TYPE	SIGNIFICANCE	NOTES
Duration	NumberSpec	Determines how long the **Function** will run once it starts.	The value can be set to a constant, a draw from a probability distribution, or the result of executing a script.
Begin Logic	ScriptSpec	Begin Logic contains a script that is executed at the very beginning of **Function** execution.	
Exit Logic	ScriptSpec	The Exit Logic specifies – through a script – which exit is to be taken for a multi-exit **Function**.	If the exit logic is nil**,** the selection probability attributes of the *exits by* relationships are used to choose the exit. If the selection attributes are not set, each exit branch will have an equal likelihood of selection. Selection of the exit occurs at the end of **Function** execution. Note, if the **Function** has only a single exit, the script will not be executed.
End Logic	ScriptSpec	End Logic contains a script that is executed at the very end of **Function** execution.	
Timeout	NumberSpec	Maximum time permitted between **Function** enablement and the start of execution.	If the timeout is exceeded, the function will not execute. Upon timeout, if a timeout exit branch (**Exit**) exists, it is automatically selected.

ATTRIBUTE	DATA TYPE	SIGNIFICANCE	NOTES
Execute Decomposition	Boolean	Indicates whether the **Function** or its decomposition is to be executed. *True* means that the **Function's** decomposition is executed. When *false* is selected, the **Function's** decomposition is ignored and the **Function's** attributes and relationships are used.	The default value for this attribute is *true*; all decomposed functions are fully expanded and executed during simulation. To execute at a higher level, one can set the Execute Decomposition to *false*. For these functions, the model will execute at that higher level and will ignore the more detailed decomposition models below it.
Acquire Available *captures*	Boolean	*True* means that the **Function** acquires the amount of **Resources** currently available and waits for the remainder of the **Resource** to become available before it executes. *False* means that the **Function** waits for the full amount to become available before it acquires the **Resource** amount it needs to execute.	*True* has the effect of making resource requests into a FIFO sequence. *False* allows a function with smaller resource needs to take the resource while the function with larger needs is waiting.
Amount *captures*	NumberSpec	This value is the amount of **Resource** that will be acquired when the **Function** starts execution and released when execution terminates.	Can be a constant, random number, or defined by a script. Amount is determined upon **Function** enablement.
Acquire Available *consumes*	Boolean	*True* means that the **Function** acquires the amount of **Resources** currently available and waits for the remainder of the **Resource** to become available before it executes. *False* means that the **Function** waits for the full amount to become available before it acquires the **Resource** amount it needs to execute.	*True* has the effect of making resource requests into a FIFO sequence. *False* allows a **Function** with smaller resource needs to take the **Resource** while the **Function** with larger needs is waiting.
Amount *consumes*	NumberSpec	This value is the amount of **Resource** that will be consumed by the **Function** when it starts execution.	Can be a constant, random number, or defined by a script. Amount is determined upon **Function** enablement.
Amount *produces*	NumberSpec	This value is the amount of **Resource** that will be produced by the **Function** when it completes execution.	Can be a constant, random number, or defined by a script. Amount is determined upon **Function** termination.
Type *exits by*	Enumeration with possible values: Normal Exception Timeout	This value identifies the type of exit from the related function. The *Timeout* exit is taken if the **Function** times out; otherwise a *Normal* or *Exception* exit is taken.	

2.3.2 Item

Items are the inputs, triggers, and outputs from **Functions**. As shown in Table 4, two attributes guide and affect simulation results.

Table 4 Item Attributes

ATTRIBUTE	DATA TYPE	SIGNIFICANCE	NOTES
Type	Enumeration with possible values: nil Analog Digital Physical Mixed	The attribute type determines how items are handled in triggering multiple **Functions**.	If the item Type is *Physical*, the Item will trigger only the first **Function** that is enabled. The others will wait for a later instance of the **Item**. Any other type is duplicated, triggering all **Functions** related to the Item via the *triggered by* relation.
Size	NumberSpec	This value identifies the size of the **Item** that is transferred by a **Link**. The total time for an **Item** to be transferred across a **Link** is: **Link** Delay + (**Item** Size / **Link** Capacity).	

2.3.3 Link

A **Link** is the physical implementation of an interface. The **Link** element represents the path over which the Items are transferred. As shown in Table 5, two key **Link** characteristics are Capacity and Delay.

Table 5 Link Attributes

ATTRIBUTE	DATA TYPE	SIGNIFICANCE	NOTES
Capacity	NumberSpec	This value identifies the flow capacity of the **Link.** Assuming an ideal model, **Item** and **Link** time Units are the same, a **Link** with a Capacity of 1.0 can transfer an **Item** of Size 1.0 in 1 simulation time unit.	
Delay	NumberSpec	Delay represents the time delay that occurs when an **Item** traverses the **Link**. It is intended to model the actual propagation delay and processing delay, not the delay due to **Resource** utilization. The total transfer time is the Delay time plus the actual transfer time of the **Item**.	

2.3.4 DomainSet

A **DomainSet** element is used to define the number of iterations in a control structure. Table 6 describes the attribute that may affect the simulator results.

Table 6 DomainSet Attributes

ATTRIBUTE	DATA TYPE	SIGNIFICANCE	NOTES
Count	NumberSpec	Count specifies the number of iterations for the associated construct.	Can be a constant, random number, or defined by a script.

2.3.5 Exit

An **Exit** element identifies a possible path to follow when a **Function** completes. By definition, multi-exit **Functions** have more than one **Exit** (i.e., more than one target of the *exits by* relation). Table 7 describes each of the attributes of the **Exit** *exit for* relationship that affects the simulator results.

Table 7 Exit Attributes

ATTRIBUTE	DATA TYPE	SIGNIFICANCE	NOTES
Type *exit for*	Enumeration with possible values: Normal Exception Timeout	This value identifies the type of **Exit** from the related **Function**. *Timeout* **Exit** is taken if the **Function** times out; otherwise, *Normal* or *Exception* **Exit** is taken.	
Selection Probability *exit for*	Float	This value indicates the likelihood of exiting the **Function** via the **Exit**.	This value is normalized, meaning that the sum of all the probabilities for all exits does not have to be 1.0. If the selection likelihood of taking **Exit** 1 is 3 and the selection likelihood of taking **Exit** 2 is 2, then the probability of the **Function** exiting by **Exit** 1 is 3/5 or 60%. NOTE: Setting a selection probability overrides use of the default (equal likelihood). Setting the Exit Logic attribute of a **Function** takes precedence over any selection probabilities.

2.3.6 Resource

The **Resource** element represents something (e.g., power, MIPs, channels, etc.) that a system uses or generates. Each **Function** may *consume*, *capture*, or *produce* **Resources**.

Table 8 describes the **Resource** attributes and the attributes of *consumed by*, *captured by*, and *produced by* relations that affect simulation results.

Table 8 Resource Attributes

ATTRIBUTE	DATA TYPE	SIGNIFICANCE	NOTES
Amount Type	Enumeration with possible values: Float Integer	Indicates the type of value for the **Resource** amounts.	
Initial Amount	NumberSpec	Specifies the amount of the **Resource** that is present in the system when the system starts.	
Maximum Amount	NumberSpec	Specifies the largest amount of the **Resource** that can be present in the system at any time.	Once the maximum is reached, the **Function** producing the **Resource** will continue to execute but will not produce any of the resource until its level drops below the maximum.
Acquire Available *captured by*	Boolean	*True* means that the **Function** acquires the amount of the **Resource** currently available and waits for the remainder of the **Resource** to become available before it executes. *False* means that the **Function** waits for the full amount to become available before it acquires the **Resource** amount it needs to execute.	***True*** has the effect of making **Resource** requests into a FIFO sequence. ***False*** allows a **Function** with smaller resource needs to take the **Resource** while the **Function** with larger needs is waiting.
Amount *captured by*	NumberSpec	Determines the amount of the **Resource** that will be acquired when the **Function** starts execution and released when the execution terminates.	Can be a constant, random number, or defined by a script. Amount is determined upon **Function** enablement.
Acquire Available *consumed by*	Boolean	*True* means that the **Function** acquires the amount of the **Resource** currently available and waits for the remainder of the **Resource** to become available before it executes. *False* means that the **Function** waits for the full amount to become available before it acquires the **Resource** amount it needs to execute.	*True* has the effect of making **Resource** requests into a FIFO sequence. *False* allows a **Function** with smaller **Resource** needs to take the **Resource** while the **Function** with larger needs is waiting.
Amount *consumed by*	NumberSpec	Determines the amount of the **Resource** to be acquired when the **Function** starts execution.	Can be a constant, random number, or defined by a script. Amount is determined upon **Function** enablement.
Amount *produced by*	NumberSpec	Determines the amount of the **Resource** to be generated upon completion of the **Function**.	Can be a constant, random value, or defined by a script. Any computation is made at the end of **Function** execution.

2.4 NumberSpec

NumberSpec is a special GENESYS attribute type that provides a great deal of flexibility in specifying and

modeling systems. A NumberSpec can be a constant value, a random value or the result of a script. NumberSpec attributes are listed in Table 9. A NumberSpec Type Selector window is shown in Figure 16.

Table 9 NumberSpec Attributes

Attribute	Class/Relationship
Duration	**Function**
Amount	**Function-Resource** *captures/captured by* relation
Amount	**Function-Resource** *consumes/consumed by* relation
Amount	**Function-Resource** *produces/produced by* relation
Timeout	**Function-Exit** *exits by/exit for* relationship
Count	**DomainSet**
Size	**Item**
Capacity	**Link**
Delay	**Link**
Initial Amount	**Resource**
Maximum Amount	**Resource**

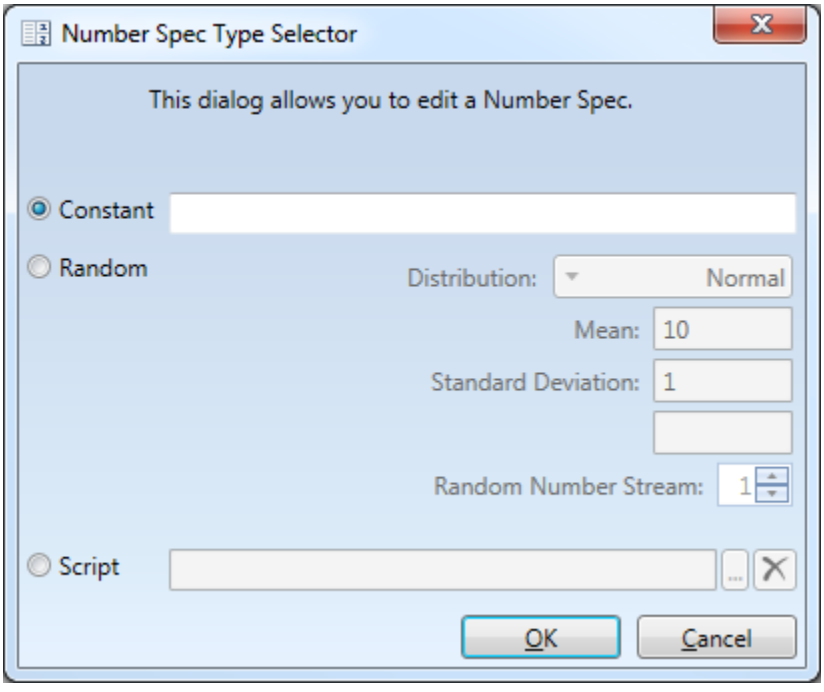


Figure 16 NumberSpec Type Selector Window

- Select *Constant* to set the attribute value to a fixed value throughout the simulation execution.
- Select *Random* to utilize a distribution to set the attribute value at the start of each execution of the

Function. There are a number of distributions from which to choose. The default distribution is the one set in ***User Random Distribution Preferences***. For a listing of the possible distributions and a brief description of each, see Appendix A – GENESYS Simulator Distributions.

- Select the *Script* option to create a Microsoft Visual Basic script to determine the attribute value. The Script Editor Window as shown in Figure 17. GENESYS will execute the script at the start of each execution of the **Function** once the necessary **Resources** have been acquired and triggers processed. More information on scripting is provided in Section 6.

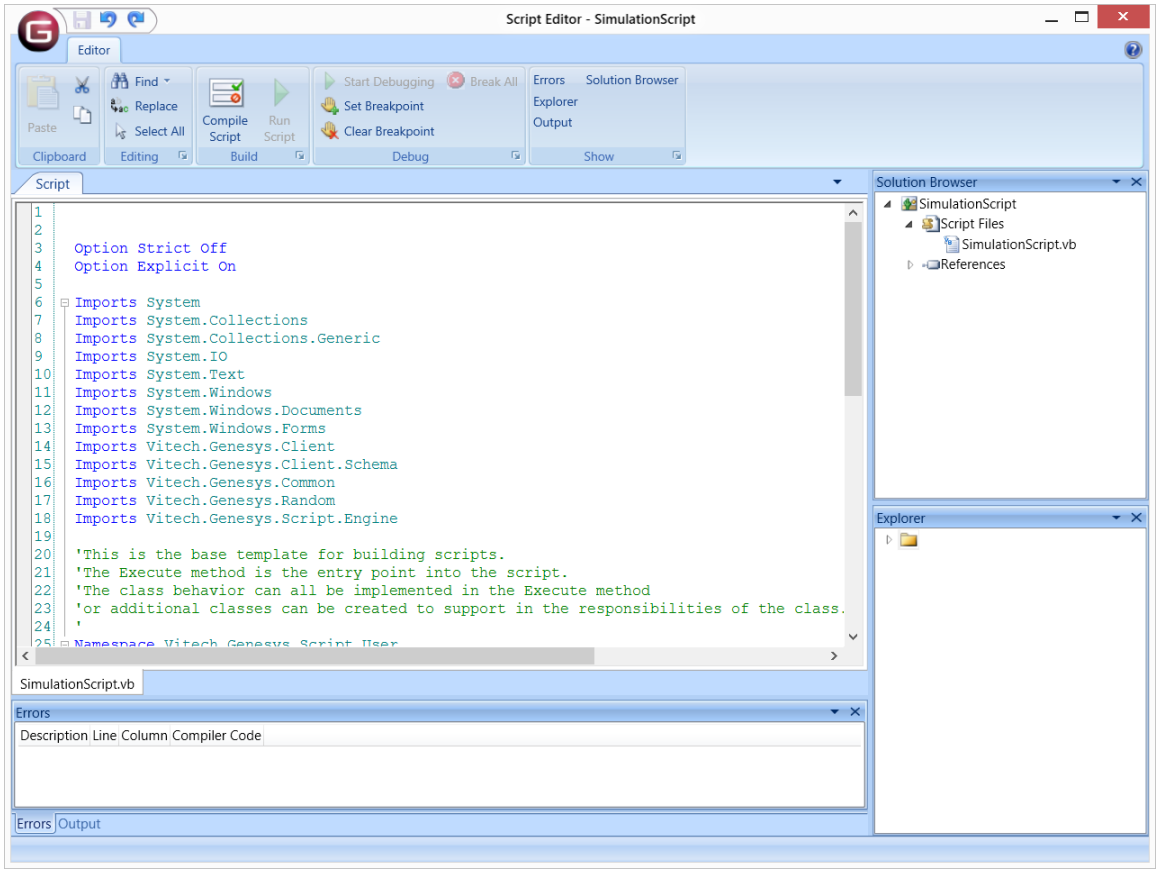


Figure 17 Script Editor Window

A NumberSpec script must exit with a Return construct that returns a variable containing the value to be used by the attribute.

2.5 ScriptSpec

A ScriptSpec is a type of attribute that is used to provide control or flexibility to a GENESYS simulation model. There are three **Function** attributes that are ScriptSpec type. These are listed in Table 10. On each execution of a given **Function**, the script defined in these attributes is executed as the **Function** is executed.

Table 10 ScriptSpec Attributes

Attribute	Class
BeginLogic	**Function**
EndLogic	**Function**
ExitLogic	**Function**

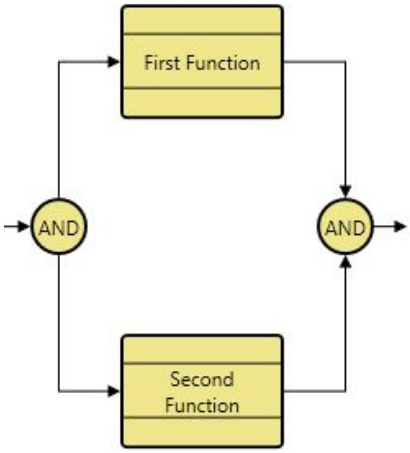
The script is entered/modified by clicking the ***Edit*** button next to the attribute in the ***Properties*** window. The ***Edit*** button opens a Script Editor window (Figure 17) where the script is built using Microsoft Visual Basic and the GENESYS API.

The last command executed in an Exit Logic ScriptSpec must be a Return construct that returns a variable containing the Exit element for the selected exit branch. There are no special requirements on Begin Logic and End Logic ScriptSpecs. Refer to Section 6 for additional information on using Microsoft Visual Basic and the GENESYS API.

3 CONTROL CONSTRUCTS

To support a full range of behaviors, the FFBD and EFFBD involve typical modeling constructs described below. It is important to understand the semantic meaning of the constructs in order to understand how they are interpreted by the GENESYS simulator.

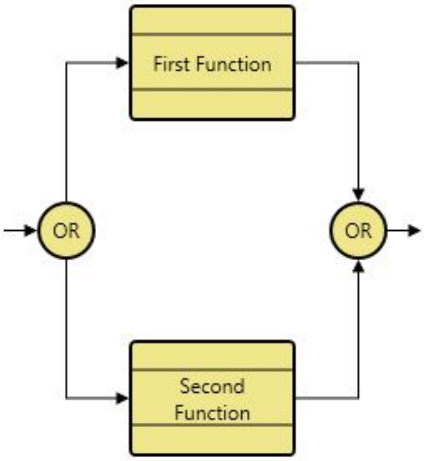
3.1 Parallel



A Parallel construct consists of an AND node, followed by separate branches that rejoin and terminate at another matching AND node. Each branch can contain any number of **Functions** and control constructs. In contrast to sequences of functions and control constructs where the next entity cannot be executed until the previous one completes, the parallel construction designates that the parallel branches can be executed concurrently (even though they may interact through triggers).

When executed by the GENESYS simulator the first entity on each branch will be enabled at the same simulation clock time. The construct cannot be exited (from the second AND node) until all branches have completed their processing. Control is then passed to the next **Function** or Construct after the parallel construct.

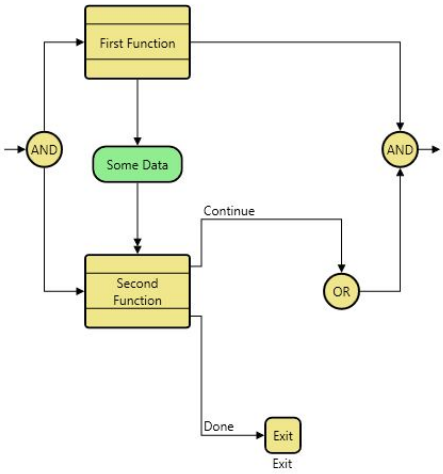
3.2 Select



A Select construct consists of an OR node followed by multiple branches that rejoin at a matching OR node. Each branch can contain any number of **Functions** and control constructs. In contrast to a parallel construct in which all branches are executed, with a Select construct only one branch is executed. Thus, the Select construct is an exclusive OR.

Each branch may be assigned a selection probability to determine the likelihood of execution during the simulation. If there are no branch selection probabilities, each branch will have equal likelihood of being selected for execution.

3.3 Multi-Exit Function



A Multi-Exit Function is a control construct where multiple branches exit from a **Function** and rejoin at an OR node. Each branch is labeled with the name of its associated **Exits** and can contain any number of **Functions** and control constructs. The criterion for selecting which exit branch to follow is specified by the **Function**, either using a Selection Probability attribute associated with the *exits by* relation or by scripting logic embedded within the Exit Logic attribute of the **Function**. If there are no exit probabilities and the Exit Logic attribute is empty, each branch will have equal likelihood of being selected.

One would normally use a Multi-Exit Function construct in lieu of a Select construct when the selection criterion can be specified by scripting logic in the Exit Logic attribute or via a corresponding exit node within the **Function's** decomposition.

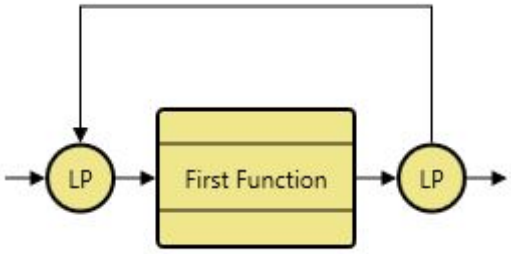
3.4 Exit Node

In the case of a Multi-Exit Function, Exit Nodes establish the mapping between the completion of the decomposition behavior and the exit branches of the parent **Function**.

There should be at least one Exit Node in the **Function** decomposition for each exit branch for the **Function**. The name for the corresponding exit branch is shown below each exit node icon.

When an Exit Node is encountered during simulation, the decomposition level is exited and the corresponding branch of the multi-exit Function is followed.

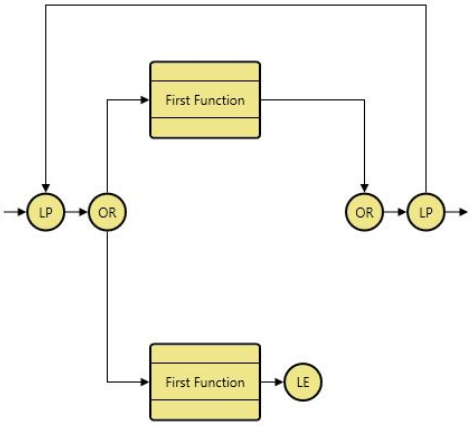
3.5 Loop



A Loop construct consists of a pair of LP nodes that enclose a branch and are connected with a loop back line. The branch can contain any number of **Functions** and control constructs. These will be repeatedly executed in sequence. The branch will typically contain a Loop Exit construct to conditionally exit the loop construct. Without a Loop Exit, the Loop construct becomes an endless loop.

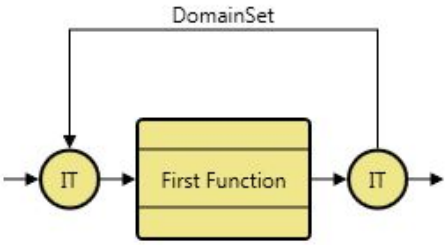
A descriptive annotation should be specified for each Loop construct. The annotation is displayed above the loop back line. This annotation is only descriptive and does not impact execution.

3.6 Loop Exit



The Loop Exit construct provides the mechanism for exiting a loop. When the Loop Exit construct is encountered, the innermost loop is immediately terminated, enabling the construct or **Function** following the loop.

3.7 Iterate

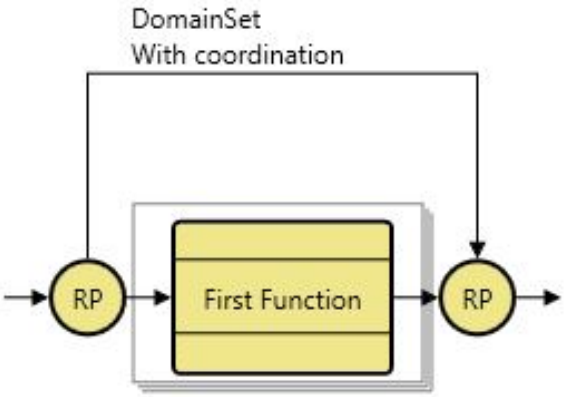


An Iterate construct consists of a pair of IT nodes that enclose a branch and are connected with a loop-back line. This line is automatically labeled with the name of the associated **DomainSet** element.

The branch can contain any number of functions and control constructs. These will be repeatedly executed (in sequence) the number of times predefined by the **DomainSet** Count attribute.

If the Count attribute of the **DomainSet** is not set, the number of iterations will revert to the system default. The default number of iterations is the one set in ***Project Simulation User Preferences***.

3.8 Replicate



A Replicate construct consists of a pair of RP nodes that enclose a main branch and are connected with a coordination branch. This coordination branch is automatically labeled with the name of the associated **DomainSet** element.

The replicate construct is a shorthand notation for identical processes that operate in parallel. The main process logic is shown on the main branch. The coordination between these processes is handled via the coordination branch. An example of a situation handled by the Replicate would be a supermarket in which multiple checkout lanes support shoppers (represented by the **Functions** on the main branch) and a manager supports the various checkout lanes as required (represented by the **Functions** on the coordination branch).

4 USING THE GENESYS SIMULATOR

The previous chapters established the building blocks of modeling that the GENESYS simulator executes. This chapter builds sample models step by step, illustrating the use of constructs and the corresponding execution characteristics.

4.1 Basic Constructs

4.1.1 Executing a Single Function

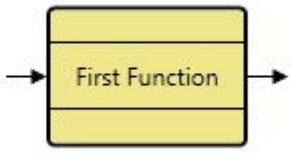
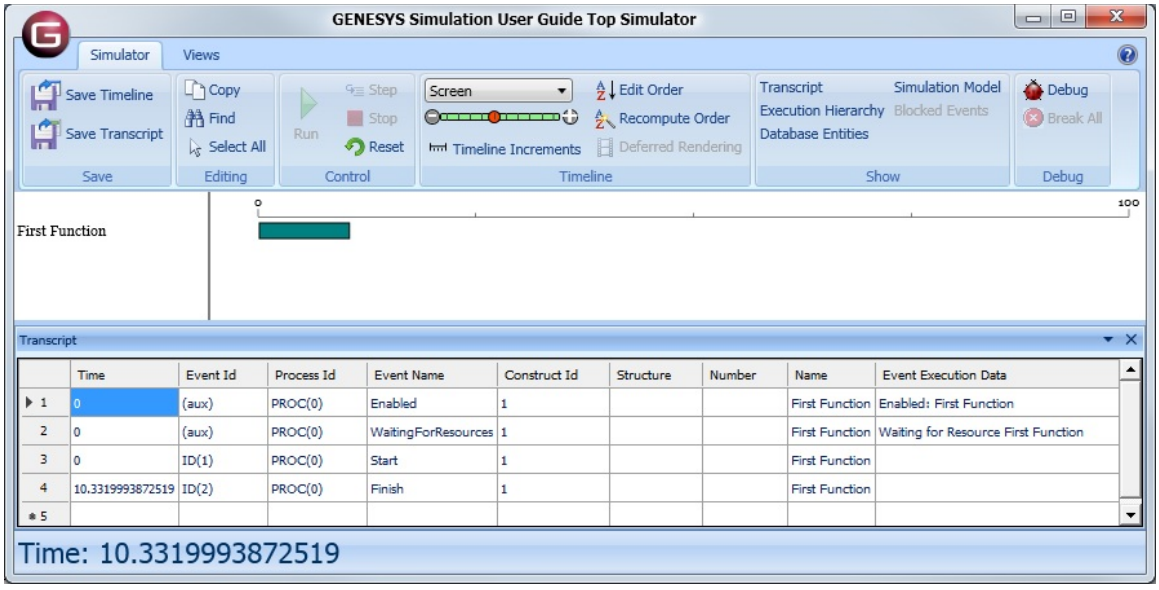


Figure 18 Single Function

Figure 18 is a model of a single **Function**. Figure 19 shows the results of executing this model. Looking at the timeline window, this **Function**, First Function, executed for 10.569 simulation time units. The timeline lists the **Function** entity in the left pane and shows an execution timeline in the right pane, demonstrating that the function execution took approximately 10 units. The transcript which appears below the timeline lists the events that occurred during this simulation run. Briefly, at time 0.0 simulation units, First Function was enabled. It was waiting for resources; there were none required, so also at time unit 0.0 the **Function** started. At time 10.5690245683168 time units, First Function completes.



	Time	Event Id	Process Id	Event Name	Construct Id	Structure	Number	Name	Event Execution Data
1	0	(aux)	PROC(0)	Enabled	1			First Function	Enabled: First Function
2	0	(aux)	PROC(0)	WaitingForResources	1			First Function	Waiting for Resource First Function
3	0	ID(1)	PROC(0)	Start	1			First Function	
4	10.3319993872519	ID(2)	PROC(0)	Finish	1			First Function	
* 5									

Figure 19 Single Function Simulation Results

Duration Default
In the Single Function example, no duration was set for the **Function**. The default preferences in GENESYS specify that in the case of a function with no duration specified, GENESYS will use a Normal distribution with a mean of 10.0 and a standard deviation of 1.0. The presence of defaults such as this enable you to quickly develop and execute models as the system evolves.

4.1.2 Executing a Sequence of Functions

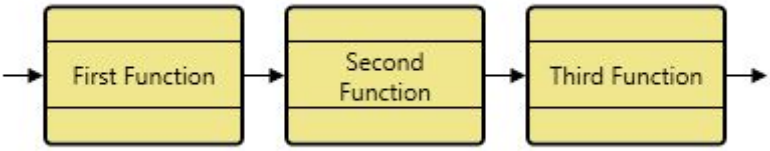
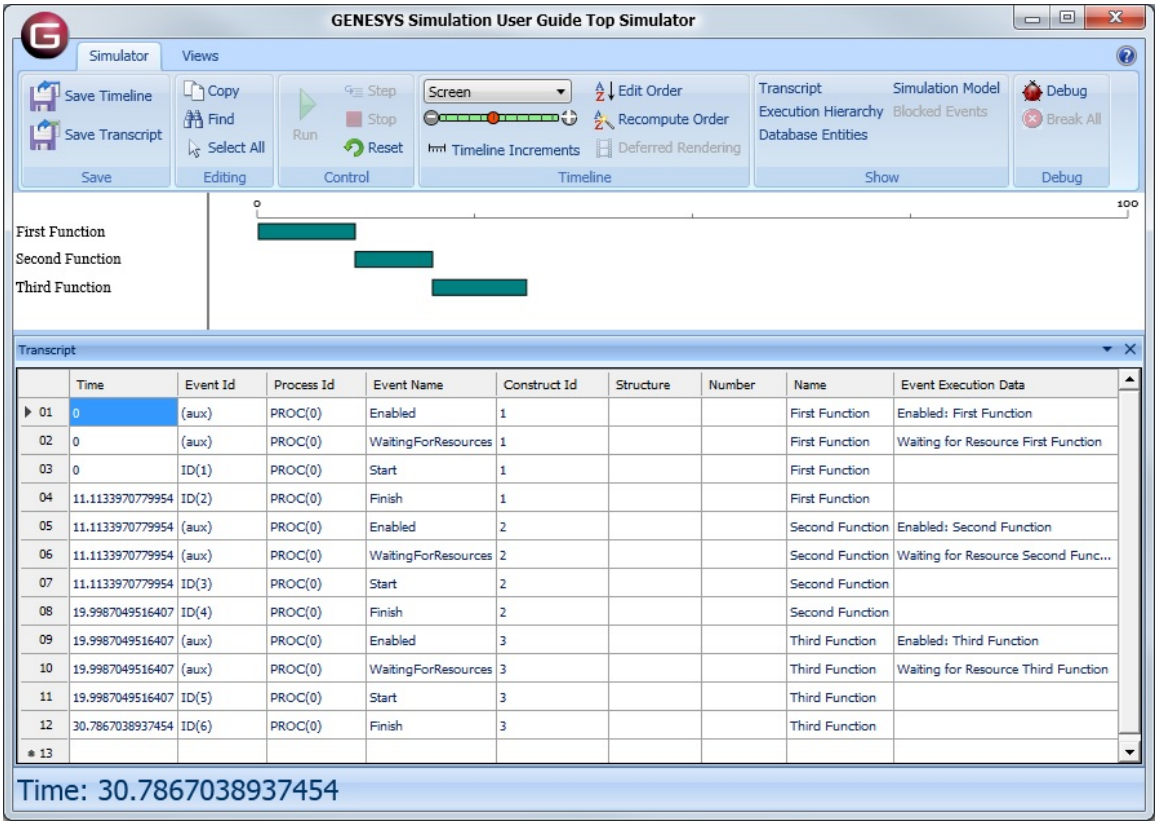


Figure 20 Sequence of Three Functions

The three **Functions** are executed one after another in sequence (Figure 20). When First Function completes, Second Function is enabled, starts, and finishes. Then it is Third Function's turn. The timeline is shown in Figure 21.



	Time	Event Id	Process Id	Event Name	Construct Id	Structure	Number	Name	Event Execution Data
01	0	(aux)	PROC(0)	Enabled	1			First Function	Enabled: First Function
02	0	(aux)	PROC(0)	WaitingForResources	1			First Function	Waiting for Resource First Function
03	0	ID(1)	PROC(0)	Start	1			First Function	
04	11.1133970779954	ID(2)	PROC(0)	Finish	1			First Function	
05	11.1133970779954	(aux)	PROC(0)	Enabled	2			Second Function	Enabled: Second Function
06	11.1133970779954	(aux)	PROC(0)	WaitingForResources	2			Second Function	Waiting for Resource Second Func...
07	11.1133970779954	ID(3)	PROC(0)	Start	2			Second Function	
08	19.9987049516407	ID(4)	PROC(0)	Finish	2			Second Function	
09	19.9987049516407	(aux)	PROC(0)	Enabled	3			Third Function	Enabled: Third Function
10	19.9987049516407	(aux)	PROC(0)	WaitingForResources	3			Third Function	Waiting for Resource Third Function
11	19.9987049516407	ID(5)	PROC(0)	Start	3			Third Function	
12	30.7867038937454	ID(6)	PROC(0)	Finish	3			Third Function	
13									

Figure 21 Multiple Functions Simulation Results

Simulator Executes the Current Behavior Model
Adding entities to the behavior model will affect the next execution of the simulator.

4.1.3 Executing a Select Construct

When a select construct is encountered, the simulator will select one branch to follow and execute all nodes and constructs on that branch. Only one branch will be taken, and by default, there is an equal chance of selecting each branch. Thus, for our example in Figure 22, there is 50/50 chance of selecting each branch.

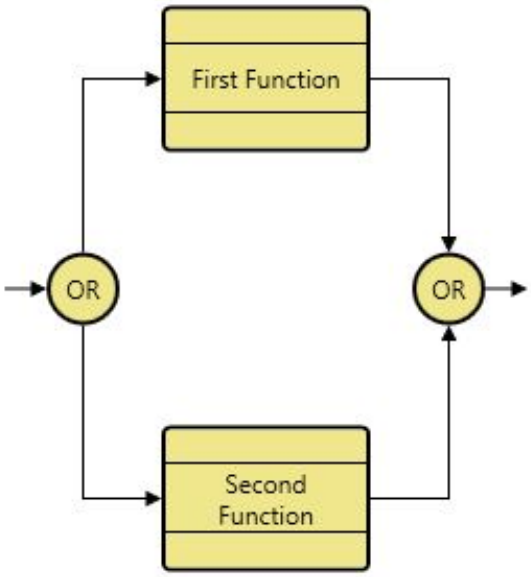
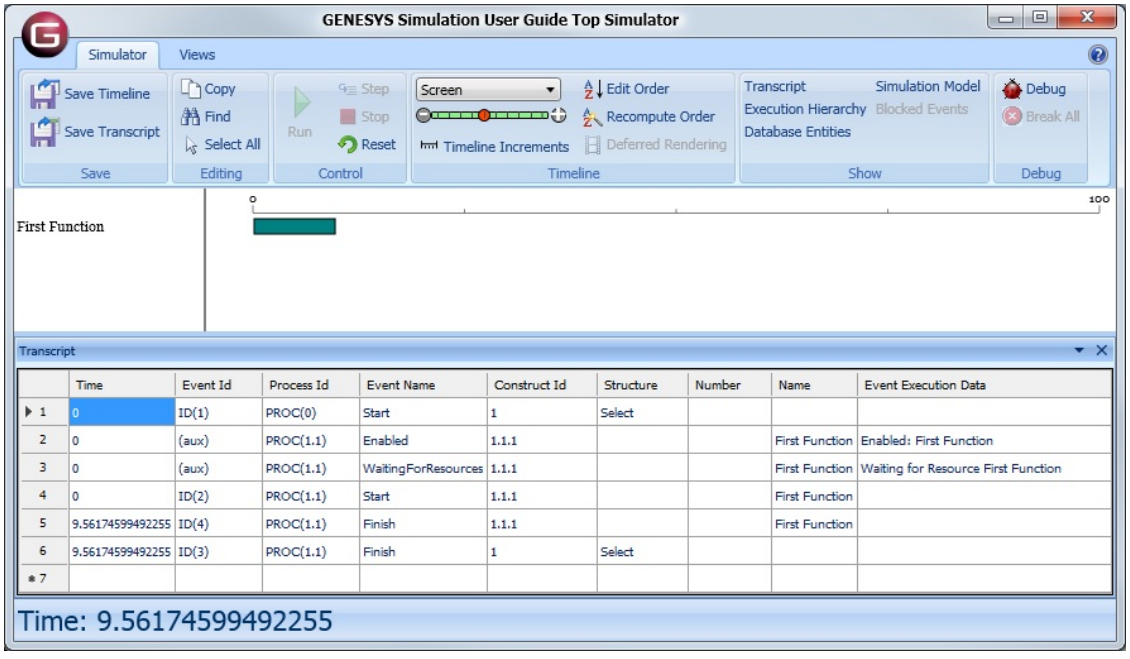


Figure 22 Select Construct

As shown in Figure 23, Second Function was executed, but if the simulator is reset and run again, then First Function may execute. There are, of course, times when one branch is weighted heavier than the other, such as when it is more likely that one function would be executed. Assigning selection probabilities is discussed in the Section 4.5, Selection Probabilities.



	Time	Event Id	Process Id	Event Name	Construct Id	Structure	Number	Name	Event Execution Data
1	0	ID(1)	PROC(0)	Start	1	Select			
2	0	(aux)	PROC(1.1)	Enabled	1.1.1			First Function	Enabled: First Function
3	0	(aux)	PROC(1.1)	WaitingForResources	1.1.1			First Function	Waiting for Resource First Function
4	0	ID(2)	PROC(1.1)	Start	1.1.1			First Function	
5	9.56174599492255	ID(4)	PROC(1.1)	Finish	1.1.1			First Function	
6	9.56174599492255	ID(3)	PROC(1.1)	Finish	1	Select			
7									

Figure 23 Select Construct Simulation Results

4.1.4 Executing a Parallel Branch

For the example model in Figure 24, the parallel construct (represented by the AND nodes) indicates both branches are to be traversed in parallel; therefore, both **Functions** are to execute concurrently. The simulator transcript and timeline in Figure 25 show that both **Function** entities were enabled and started at exactly the same time. First Function happened to finish before Second Function, but if the simulator is reset and run again, Second Function may complete first. This is once again due to the default setting for the Duration attribute, which is set to be a normal distribution. If one or both of the **Functions'** Duration attribute(s) are set to a constant value, then the **Function** will run for exactly that number of time units.

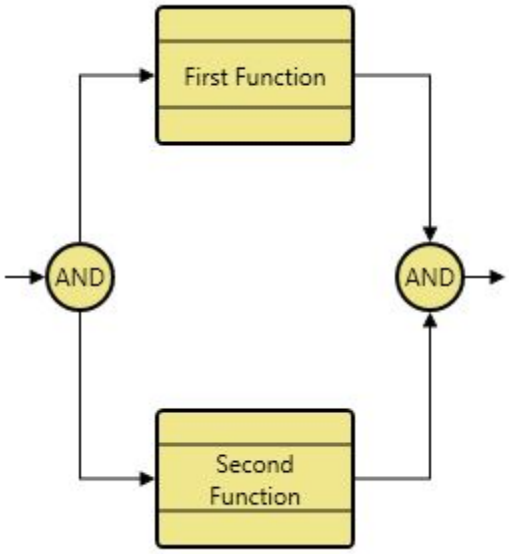
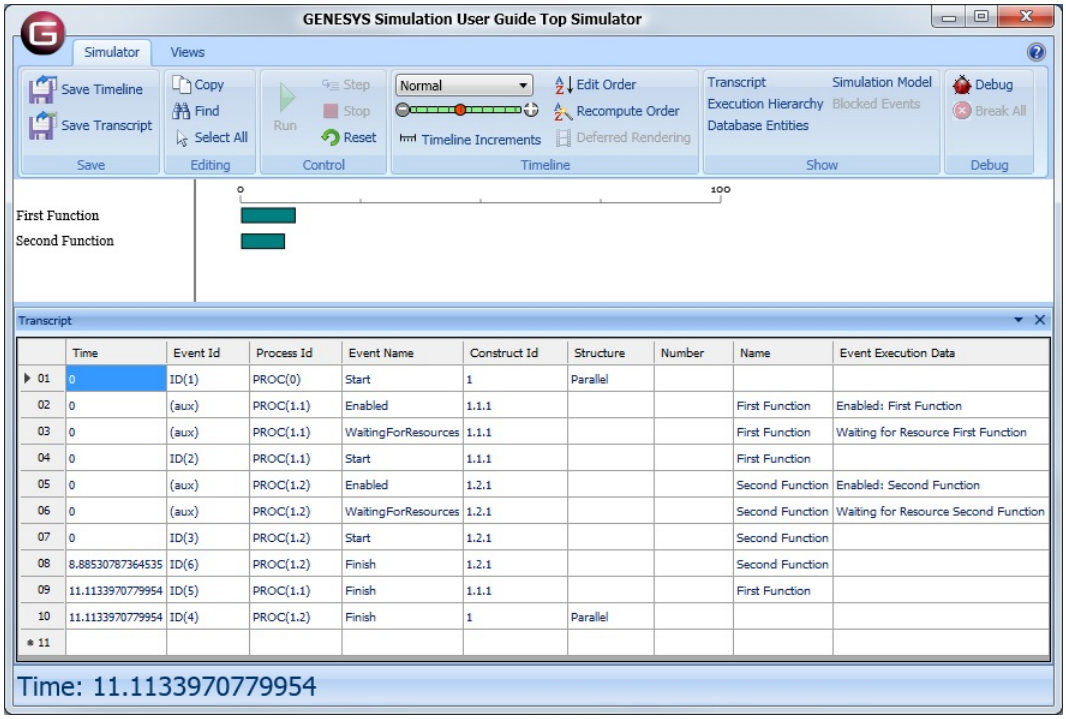


Figure 24 Parallel Construct



	Time	Event Id	Process Id	Event Name	Construct Id	Structure	Number	Name	Event Execution Data
01	0	ID(1)	PROC(0)	Start	1	Parallel			
02	0	(aux)	PROC(1.1)	Enabled	1.1.1			First Function	Enabled: First Function
03	0	(aux)	PROC(1.1)	WaitingForResources	1.1.1			First Function	Waiting for Resource First Function
04	0	ID(2)	PROC(1.1)	Start	1.1.1			First Function	
05	0	(aux)	PROC(1.2)	Enabled	1.2.1			Second Function	Enabled: Second Function
06	0	(aux)	PROC(1.2)	WaitingForResources	1.2.1			Second Function	Waiting for Resource Second Function
07	0	ID(3)	PROC(1.2)	Start	1.2.1			Second Function	
08	8.88530787364535	ID(6)	PROC(1.2)	Finish	1.2.1			Second Function	
09	11.1133970779954	ID(5)	PROC(1.1)	Finish	1.1.1			First Function	
10	11.1133970779954	ID(4)	PROC(1.2)	Finish	1	Parallel			
* 11									

Figure 25 Parallel Construct Simulation Results

4.2 Items in Models

4.2.1 Executing Data Store Items

Data store **Items** have nothing to do with the control of the model; they simply represent data that one **Function** outputs and another **Function** receives as input. They have no direct impact on the execution of the model. An example model is shown in Figure 26.

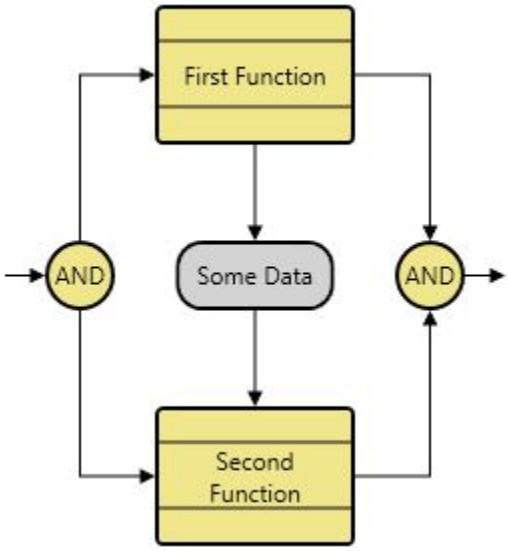
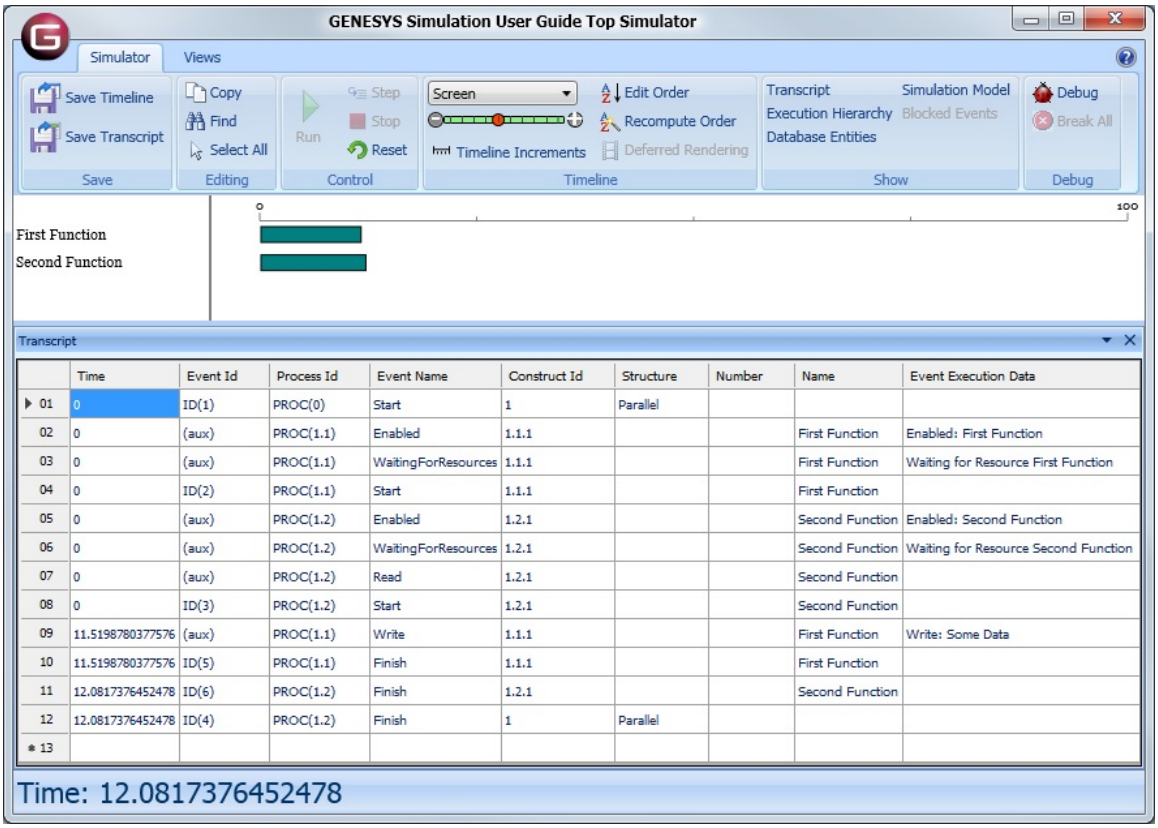


Figure 26 A Data Store Item

As shown in the GENESYS simulator transcript and timeline in Figure 27 the data store, Some Data, was written at approximately 10.08 simulation units, just as First Function and the Parallel construct were finished.



	Time	Event Id	Process Id	Event Name	Construct Id	Structure	Number	Name	Event Execution Data
01	0	ID(1)	PROC(0)	Start	1	Parallel			
02	0	(aux)	PROC(1.1)	Enabled	1.1.1			First Function	Enabled: First Function
03	0	(aux)	PROC(1.1)	WaitingForResources	1.1.1			First Function	Waiting for Resource First Function
04	0	ID(2)	PROC(1.1)	Start	1.1.1			First Function	
05	0	(aux)	PROC(1.2)	Enabled	1.2.1			Second Function	Enabled: Second Function
06	0	(aux)	PROC(1.2)	WaitingForResources	1.2.1			Second Function	Waiting for Resource Second Function
07	0	(aux)	PROC(1.2)	Read	1.2.1			Second Function	
08	0	ID(3)	PROC(1.2)	Start	1.2.1			Second Function	
09	11.5198780377576	(aux)	PROC(1.1)	Write	1.1.1			First Function	Write: Some Data
10	11.5198780377576	ID(5)	PROC(1.1)	Finish	1.1.1			First Function	
11	12.0817376452478	ID(6)	PROC(1.2)	Finish	1.2.1			Second Function	
12	12.0817376452478	ID(4)	PROC(1.2)	Finish	1	Parallel			
13									

Figure 27 Data Store Item Simulation Results

4.2.2 Executing Data Store Triggers

Triggering **Items** have an impact on control of the model. **Functions** do not start until they are enabled and receive their triggering **Item**(s).

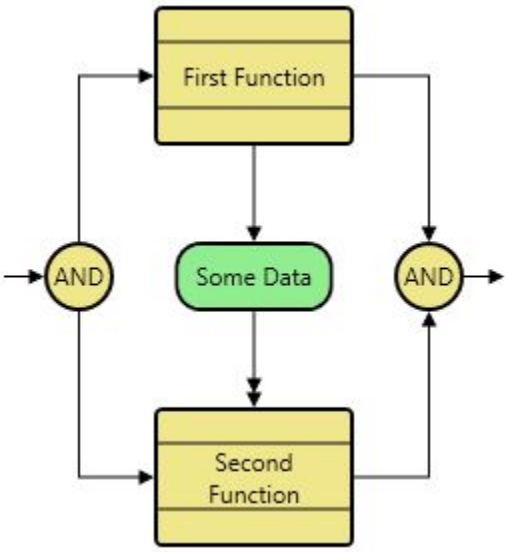
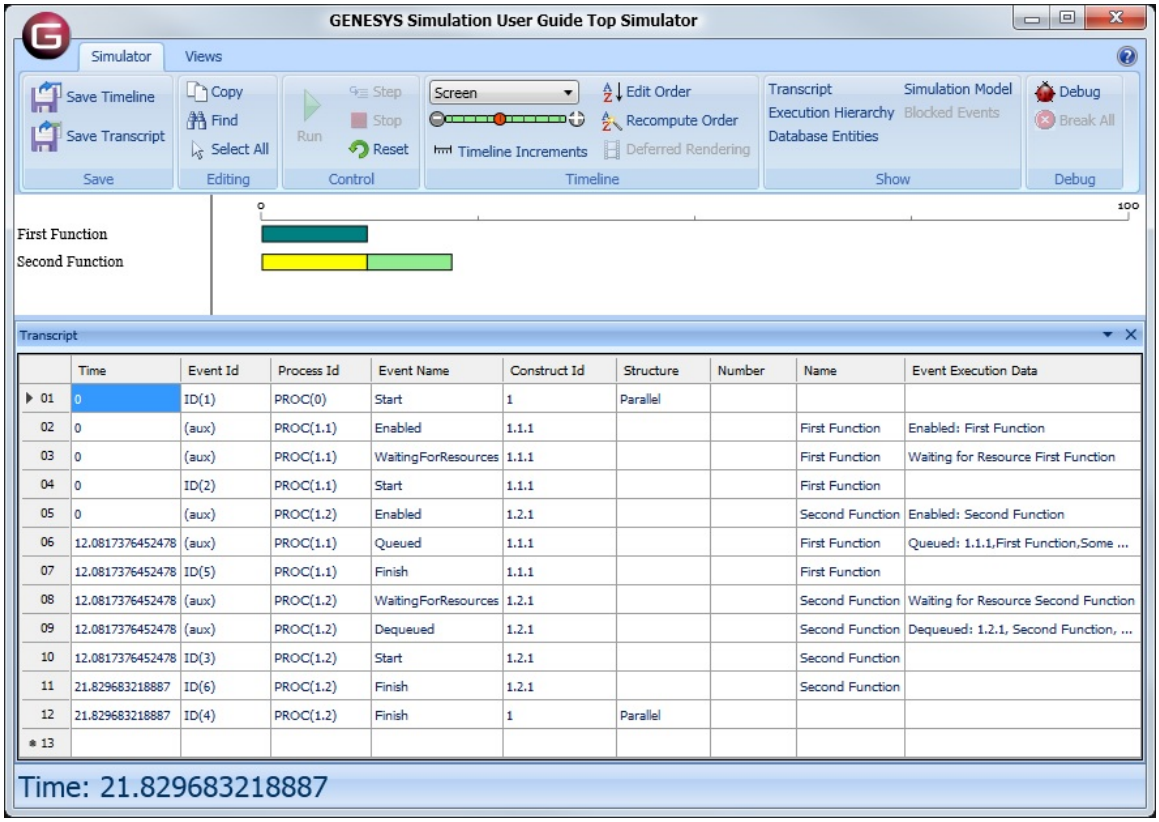


Figure 28 A Data Trigger

Earlier, when a model with a parallel branch was executed, both **Functions** began at exactly the same time. By adding a trigger between the **Functions** in Figure 28, Second Function cannot start until the trigger is received. Since the trigger is an output of First Function, it is not available until First Function is completed.



	Time	Event Id	Process Id	Event Name	Construct Id	Structure	Number	Name	Event Execution Data
01	0	ID(1)	PROC(0)	Start	1	Parallel			
02	0	(aux)	PROC(1.1)	Enabled	1.1.1			First Function	Enabled: First Function
03	0	(aux)	PROC(1.1)	WaitingForResources	1.1.1			First Function	Waiting for Resource First Function
04	0	ID(2)	PROC(1.1)	Start	1.1.1			First Function	
05	0	(aux)	PROC(1.2)	Enabled	1.2.1			Second Function	Enabled: Second Function
06	12.0817376452478	(aux)	PROC(1.1)	Queued	1.1.1			First Function	Queued: 1.1.1,First Function,Some ...
07	12.0817376452478	ID(5)	PROC(1.1)	Finish	1.1.1			First Function	
08	12.0817376452478	(aux)	PROC(1.2)	WaitingForResources	1.2.1			Second Function	Waiting for Resource Second Function
09	12.0817376452478	(aux)	PROC(1.2)	Dequeued	1.2.1			Second Function	Dequeued: 1.2.1, Second Function, ...
10	12.0817376452478	ID(3)	PROC(1.2)	Start	1.2.1			Second Function	
11	21.829683218887	ID(6)	PROC(1.2)	Finish	1.2.1			Second Function	
12	21.829683218887	ID(4)	PROC(1.2)	Finish	1	Parallel			
13									

Figure 29 Trigger Item Simulation Results

As shown above in Figure 29, Second Function was enabled at 0.0, the same time as First Function. However, it was not able to start until First Function was complete and could pass the **Item** to Second Function. As shown previously in Table 1, the timeline is color-coded to assist in this interpretation. Durations for **Functions** that do not have triggers are shown in teal. Durations for **Functions** that have triggers are shown in bright green. Yellow indicates the time span from when the **Function** was enabled to when the triggering **Item** arrived and the **Function** executed.

Displaying Items in the Timeline
If triggering Items do not appear in the timeline, select: ***Database Entities,*** in the simulator window ribbon, to display the items. Refer to Database Entities Pane (Section 1.6) for more information.

4.3 Iterations

So far, we have been looking at simple functional flows that execute once. Often functions and logic need to operate repetitively. We designate this by enclosing the repeating logic with an iterate construct.

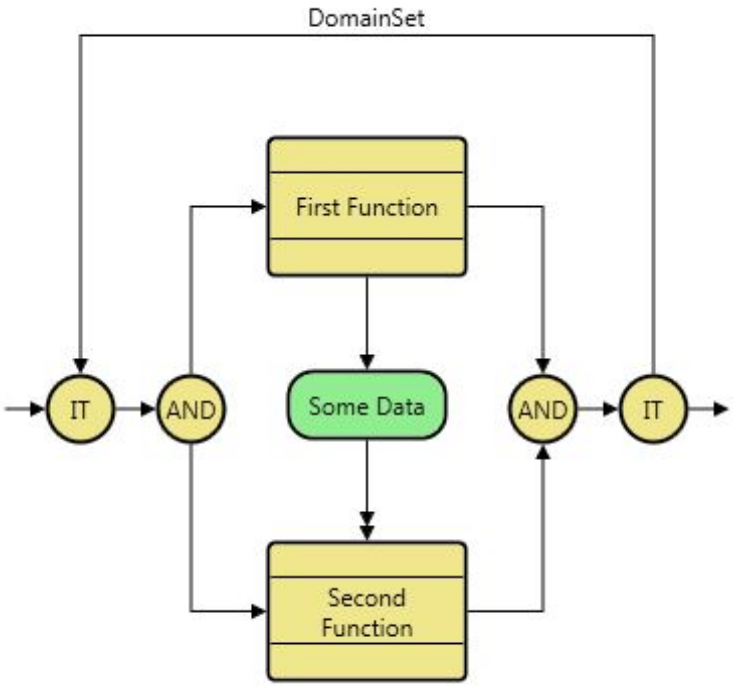
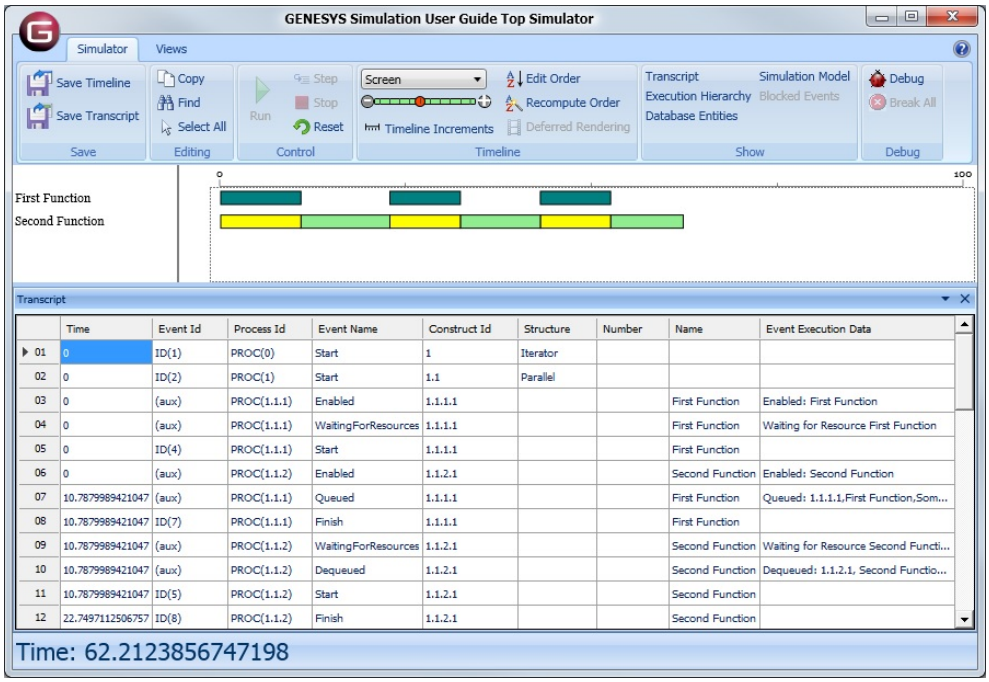


Figure 30 An Iterate Construct

When the simulator is executed on the model shown in Figure 30, the results (Figure 31) show that the model is executed three times (as specified by either the system default setting or by the **DomainSet** defining the iterate). With each iteration, both **Functions** are enabled at the same time, First Function executing immediately and Second Function waiting for the trigger **Item** before it is able to execute.



	Time	Event Id	Process Id	Event Name	Construct Id	Structure	Number	Name	Event Execution Data
01	0	ID(1)	PROC(0)	Start	1	Iterator			
02	0	ID(2)	PROC(1)	Start	1.1	Parallel			
03	0	(aux)	PROC(1.1.1)	Enabled	1.1.1.1			First Function	Enabled: First Function
04	0	(aux)	PROC(1.1.1)	WaitingForResources	1.1.1.1			First Function	Waiting for Resource First Function
05	0	ID(4)	PROC(1.1.1)	Start	1.1.1.1			First Function	
06	0	(aux)	PROC(1.1.2)	Enabled	1.1.2.1			Second Function	Enabled: Second Function
07	10.7879989421047	(aux)	PROC(1.1.1)	Queued	1.1.1.1			First Function	Queued: 1.1.1.1,First Function,Som...
08	10.7879989421047	ID(7)	PROC(1.1.1)	Finish	1.1.1.1			First Function	
09	10.7879989421047	(aux)	PROC(1.1.2)	WaitingForResources	1.1.2.1			Second Function	Waiting for Resource Second Functi...
10	10.7879989421047	(aux)	PROC(1.1.2)	Dequeued	1.1.2.1			Second Function	Dequeued: 1.1.2.1, Second Functio...
11	10.7879989421047	ID(5)	PROC(1.1.2)	Start	1.1.2.1			Second Function	
12	22.7497112506757	ID(8)	PROC(1.1.2)	Finish	1.1.2.1			Second Function	

Figure 31 Iterate Construct Simulation Results

4.4 Multi-Exit Functions

Up to this point, we have seen basic simulation models where one **Function** progresses to the next. In reality, a **Function** may have multiple exit paths that represent a selection of the next steps to be taken given some condition identified by the **Function**. In Figure 32 we added multiple exits (i.e., by establishing three targets for the *exits by* relation) from First Function, thus adding the following three new **Exit** elements: Bad Application, Incomplete, and Good Application. On each exit branch, we added a **Function** (Fix Application, Do More Work, and Next Function, respectively). Depending on the results of First Function, the execution will continue with *Fix Application, Do More Work,* or *Next Action* after the **Item** (Some Data) is output and we are able to trigger Our Second Function.

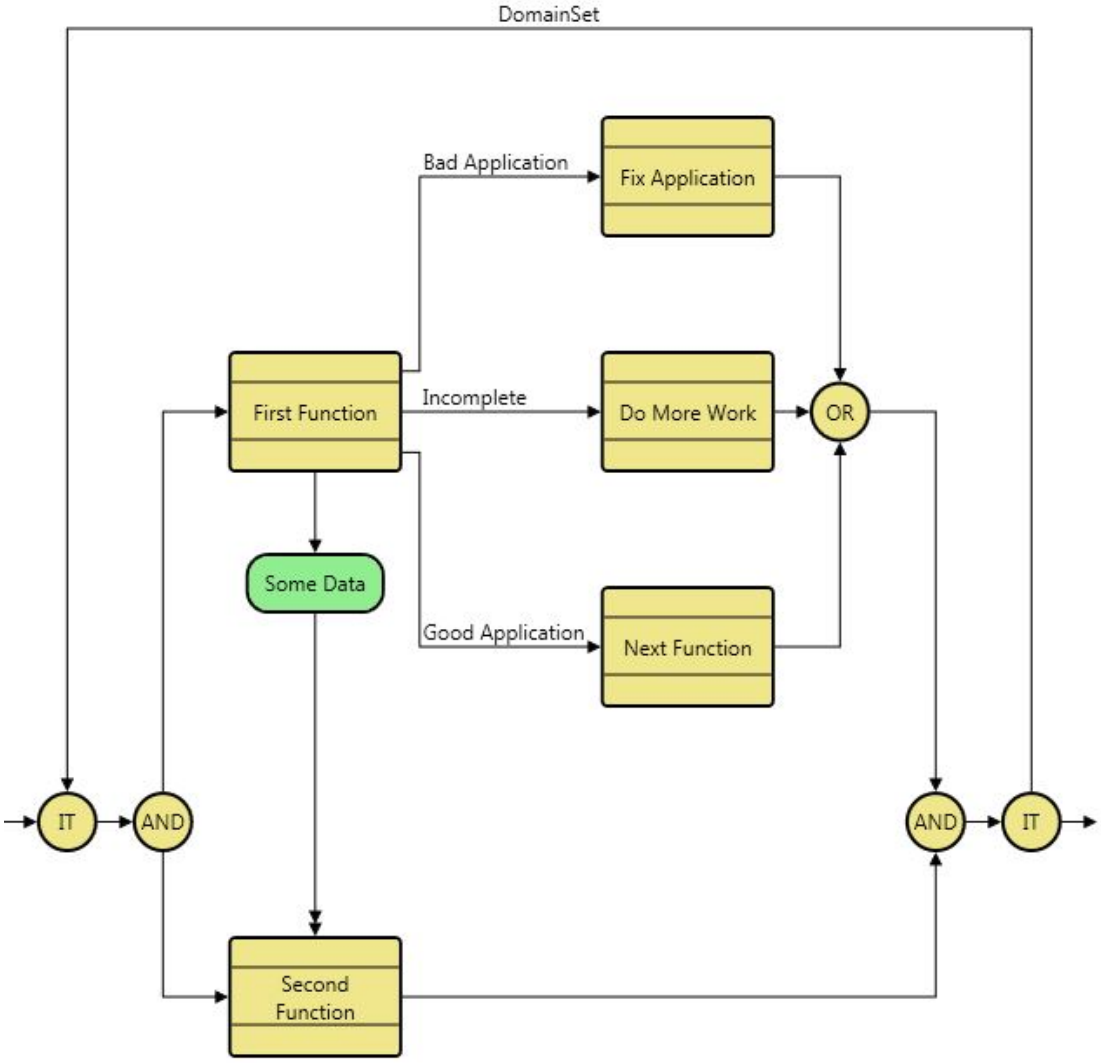
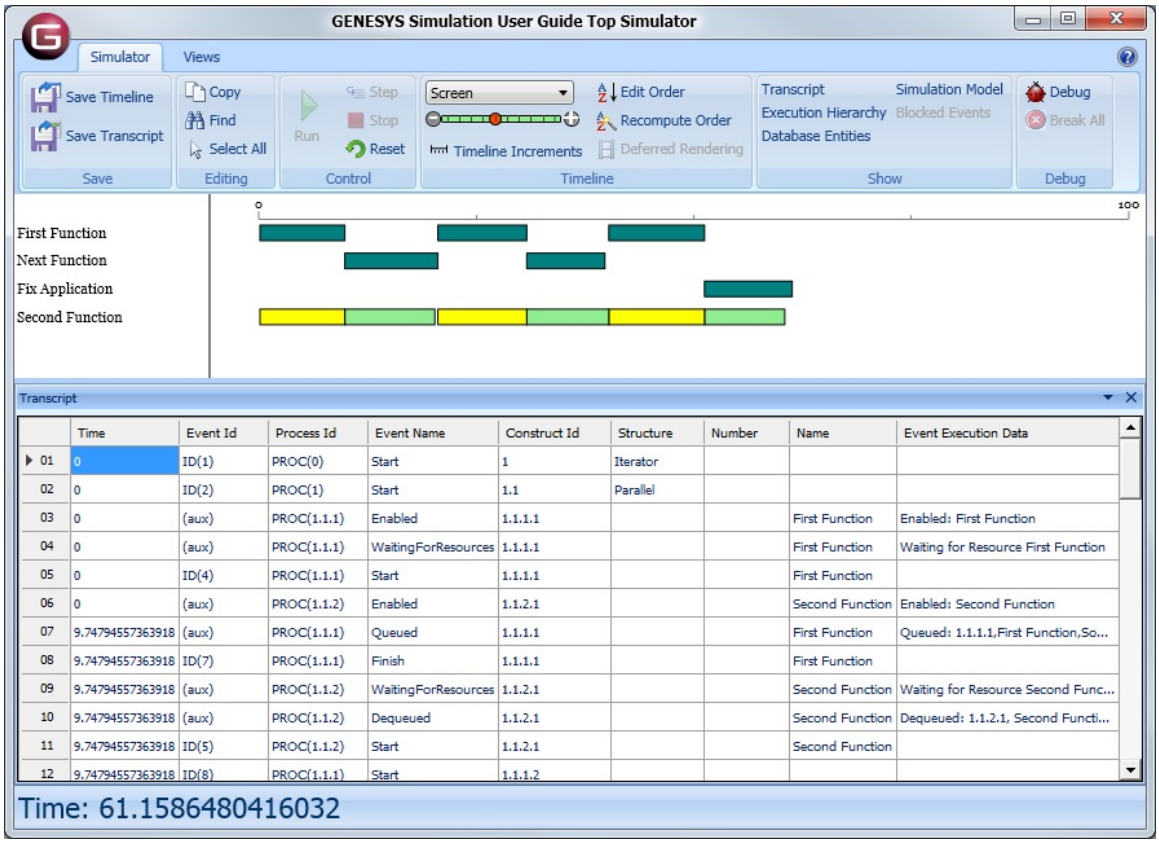


Figure 32 A Multi-Exit Function Construct

When the simulator is run on this model, the simulator generates a random number and selects the appropriate branch. Note from the resulting timeline (Figure 33) that First Function executed each time the simulator looped through the Iterate. Since First Function has three exit branches, the **Functions** performed following First Function will be dependent on the random number generator used by the simulator. In the first iteration, First Function *exits by* Good Application (resulting in the execution of Next Function). In the second iteration, First Functions *exits by* Bad Application (resulting in the execution of Fix Application). In the third iteration, First Function also *exits by* Good Application. Note in all three cases, Second Function

awaits the trigger output from First Function before beginning execution. If we reset and run again, our results will vary since we are using a random number generator to select the exit branch.



	Time	Event Id	Process Id	Event Name	Construct Id	Structure	Number	Name	Event Execution Data
01	0	ID(1)	PROC(0)	Start	1	Iterator			
02	0	ID(2)	PROC(1)	Start	1.1	Parallel			
03	0	(aux)	PROC(1.1.1)	Enabled	1.1.1.1			First Function	Enabled: First Function
04	0	(aux)	PROC(1.1.1)	WaitingForResources	1.1.1.1			First Function	Waiting for Resource First Function
05	0	ID(4)	PROC(1.1.1)	Start	1.1.1.1			First Function	
06	0	(aux)	PROC(1.1.2)	Enabled	1.1.2.1			Second Function	Enabled: Second Function
07	9.74794557363918	(aux)	PROC(1.1.1)	Queued	1.1.1.1			First Function	Queued: 1.1.1.1,First Function,So...
08	9.74794557363918	ID(7)	PROC(1.1.1)	Finish	1.1.1.1			First Function	
09	9.74794557363918	(aux)	PROC(1.1.2)	WaitingForResources	1.1.2.1			Second Function	Waiting for Resource Second Func...
10	9.74794557363918	(aux)	PROC(1.1.2)	Dequeued	1.1.2.1			Second Function	Dequeued: 1.1.2.1, Second Functi...
11	9.74794557363918	ID(5)	PROC(1.1.2)	Start	1.1.2.1			Second Function	
12	9.74794557363918	ID(8)	PROC(1.1.1)	Start	1.1.1.2				

Figure 33 Multi-Exit Function Construct Simulation Results

4.5 Selection Probabilities

The user can specify the probability that a specific branch is taken from a Select construct or that a specific exit is taken from a multi-exit **Function**. If not specified, there is an equal chance that a particular branch will be taken.

> **Note**
>
> There are no constraints on the value entered for a selection probability. If the path selection information is based on trial data or other information, it is not necessary to convert the data into percentages. GENESYS will normalize the information provided. For example, in a 2-branch select construct, if the first branch is taken 4 times and the second branch is taken 17 times and these values are entered as selection probabilities, during execution there will be a 4/21 (or 19%) chance of taking the first path.
>
> If only some branches have selection probabilities, any branch without a probability will be treated as having zero probability.

In the case of a multi-exit **Function**, one of the exit paths is selected, and execution continues down that path. The exit branches, Bad Application, Incomplete, Good Application, in our model have been assigned likelihoods of 56, 32, and 12, respectively. The sums of the assigned probability values do not have to equal 100 percent; they will be normalized by the simulator. We changed the Count attribute of the **DomainSet**, the Iterate, to 10. Figure 34 shows our revised model. Figure 35 shows the results from the simulation run.

Based on these probabilities, one would expect Fix Application to be selected most often and Next Application to be selected least, which was the case.

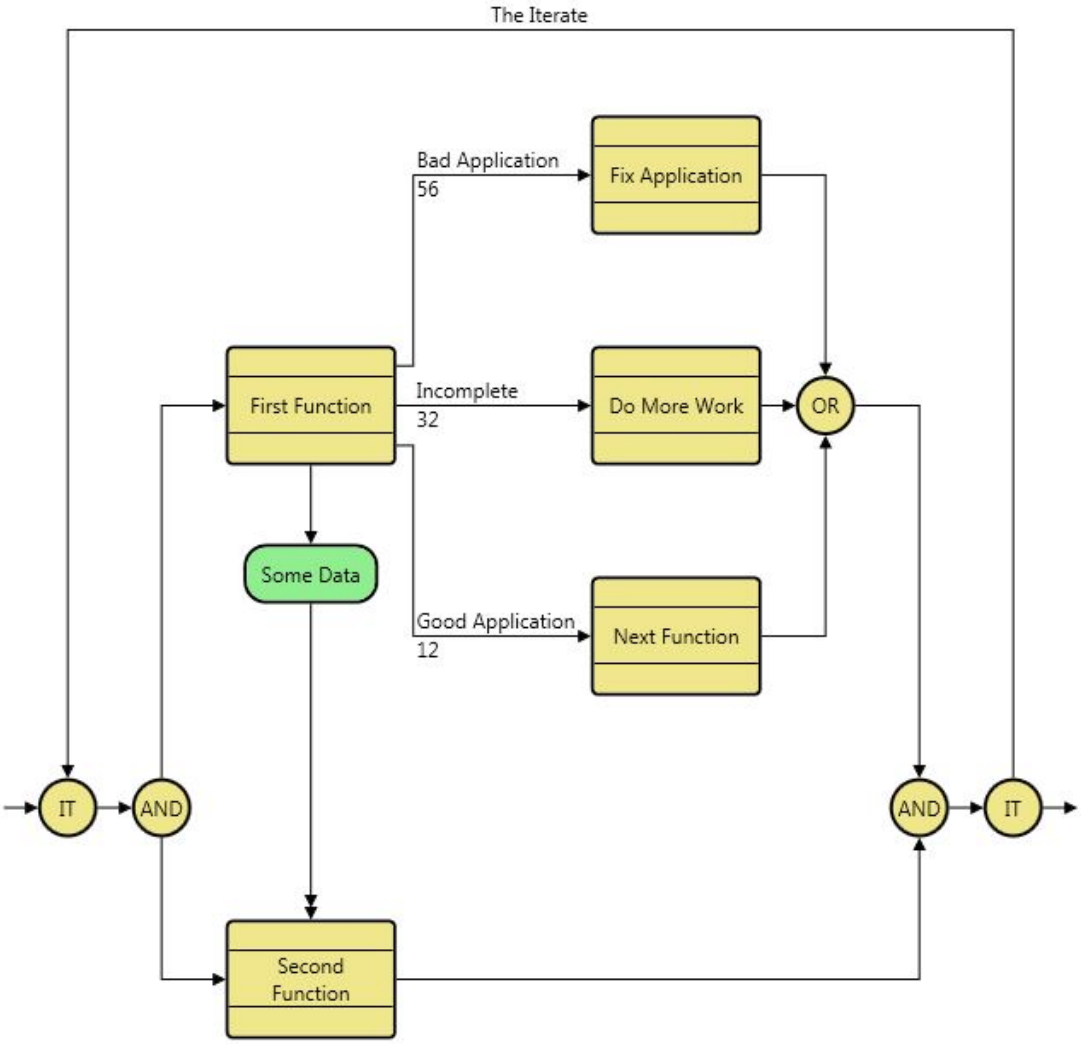
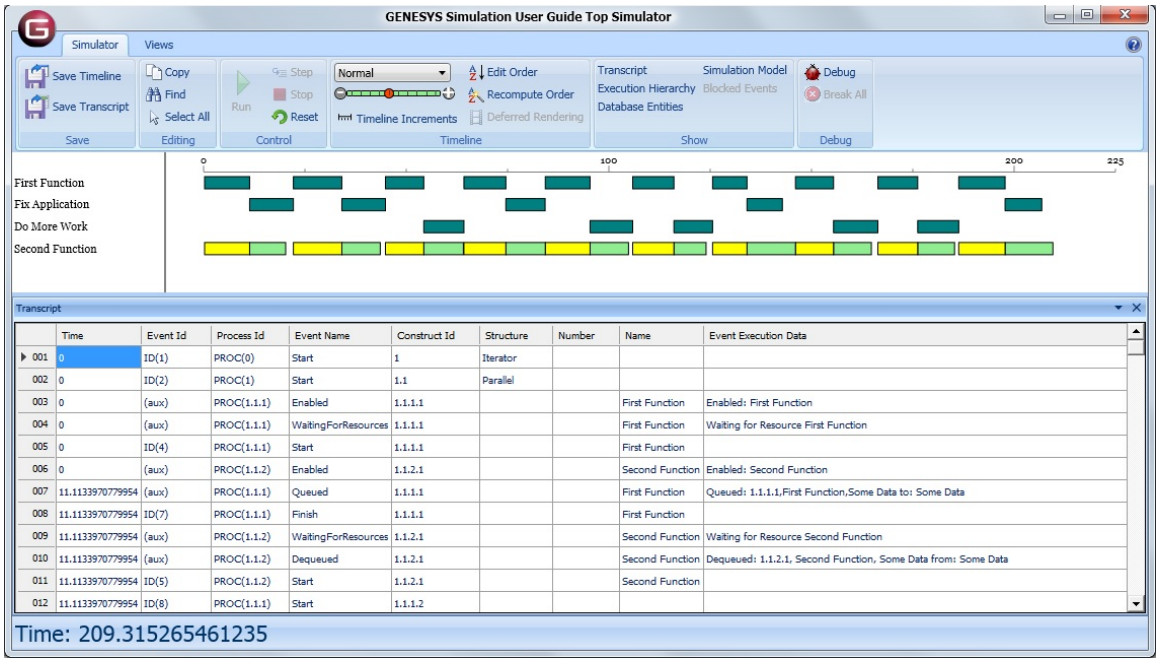


Figure 34 Multi-Exit Function Construct with Selection Probabilities



	Time	Event Id	Process Id	Event Name	Construct Id	Structure	Number	Name	Event Execution Data
001	0	ID(1)	PROC(0)	Start	1	Iterator			
002	0	ID(2)	PROC(1)	Start	1.1	Parallel			
003	0	(aux)	PROC(1.1.1)	Enabled	1.1.1.1			First Function	Enabled: First Function
004	0	(aux)	PROC(1.1.1)	WaitingForResources	1.1.1.1			First Function	Waiting for Resource First Function
005	0	ID(4)	PROC(1.1.1)	Start	1.1.1.1			First Function	
006	0	(aux)	PROC(1.1.2)	Enabled	1.1.2.1			Second Function	Enabled: Second Function
007	11.1133970779954	(aux)	PROC(1.1.1)	Queued	1.1.1.1			First Function	Queued: 1.1.1.1,First Function,Some Data to: Some Data
008	11.1133970779954	ID(7)	PROC(1.1.1)	Finish	1.1.1.1			First Function	
009	11.1133970779954	(aux)	PROC(1.1.2)	WaitingForResources	1.1.2.1			Second Function	Waiting for Resource Second Function
010	11.1133970779954	(aux)	PROC(1.1.2)	Dequeued	1.1.2.1			Second Function	Dequeued: 1.1.2.1, Second Function, Some Data from: Some Data
011	11.1133970779954	ID(5)	PROC(1.1.2)	Start	1.1.2.1			Second Function	
012	11.1133970779954	ID(8)	PROC(1.1.1)	Start	1.1.1.2				

Figure 35 Multi-Exit Function with Selection Probabilities Simulation Results

4.6 Multiple Levels

Functions may be decomposed into lower-level models. To illustrate this point, we have added application-processing functions as shown in Figure 36 as the decomposition of Second Function.

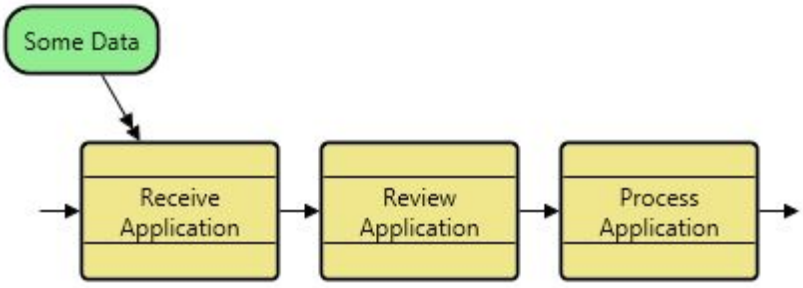


Figure 36 Second Function Decomposed

Returning to our Top Function (Figure 37), we see that once a **Function** has been decomposed to another level of activity, the **Function** icon contains a black square in the upper left corner. During simulation, if the attribute Execute Decomposition of Second Function is set to *true*, the characteristics of Second Function are ignored and the characteristics of the three subordinate functions that decompose Second Function are used. Similarly, if the attribute Execute Decomposition of Second Function is set to *false*, the characteristics of Second Function are used during execution. We have inserted another **Function** following Second Function; Process Order executes after Second Function.

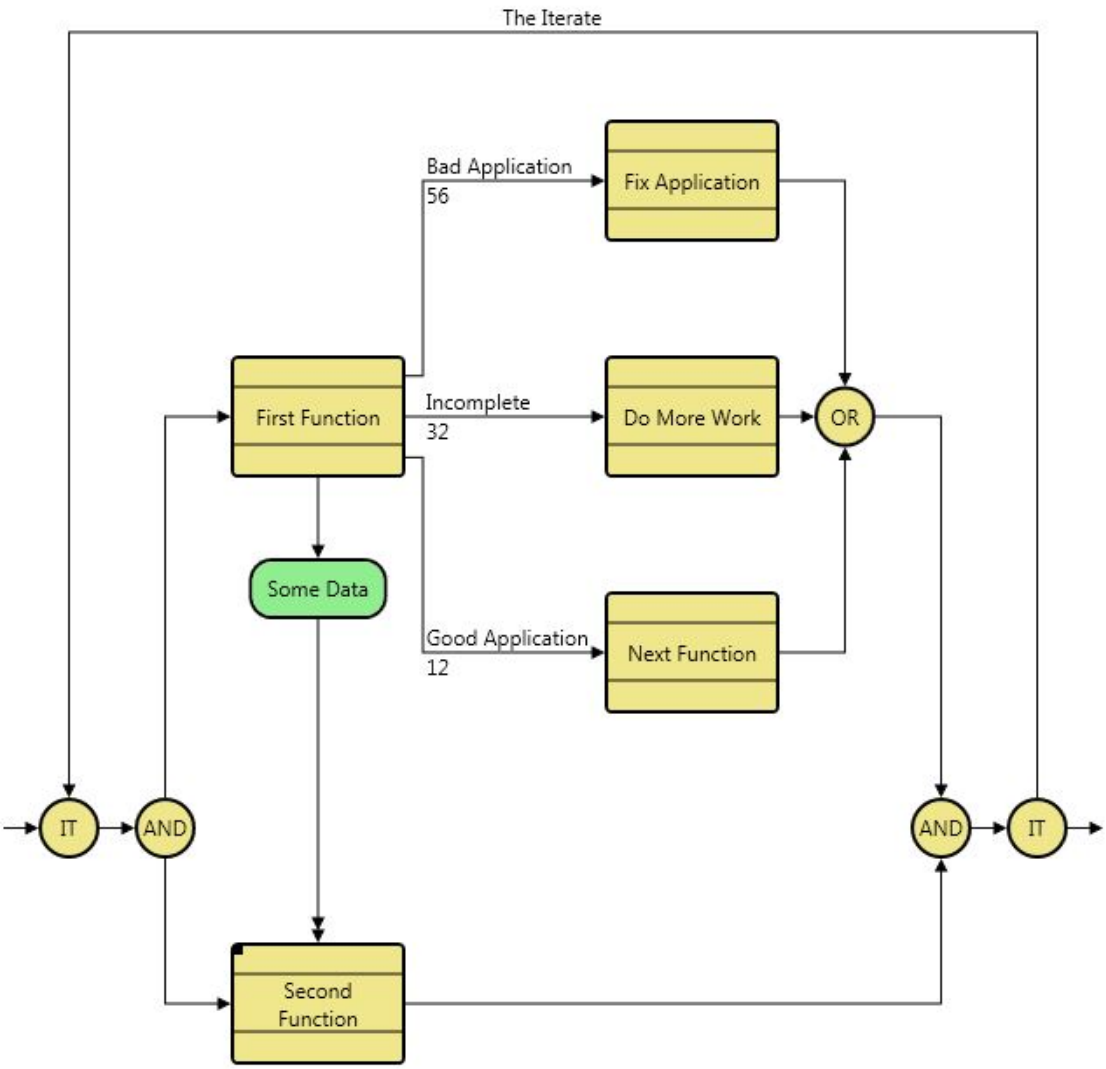
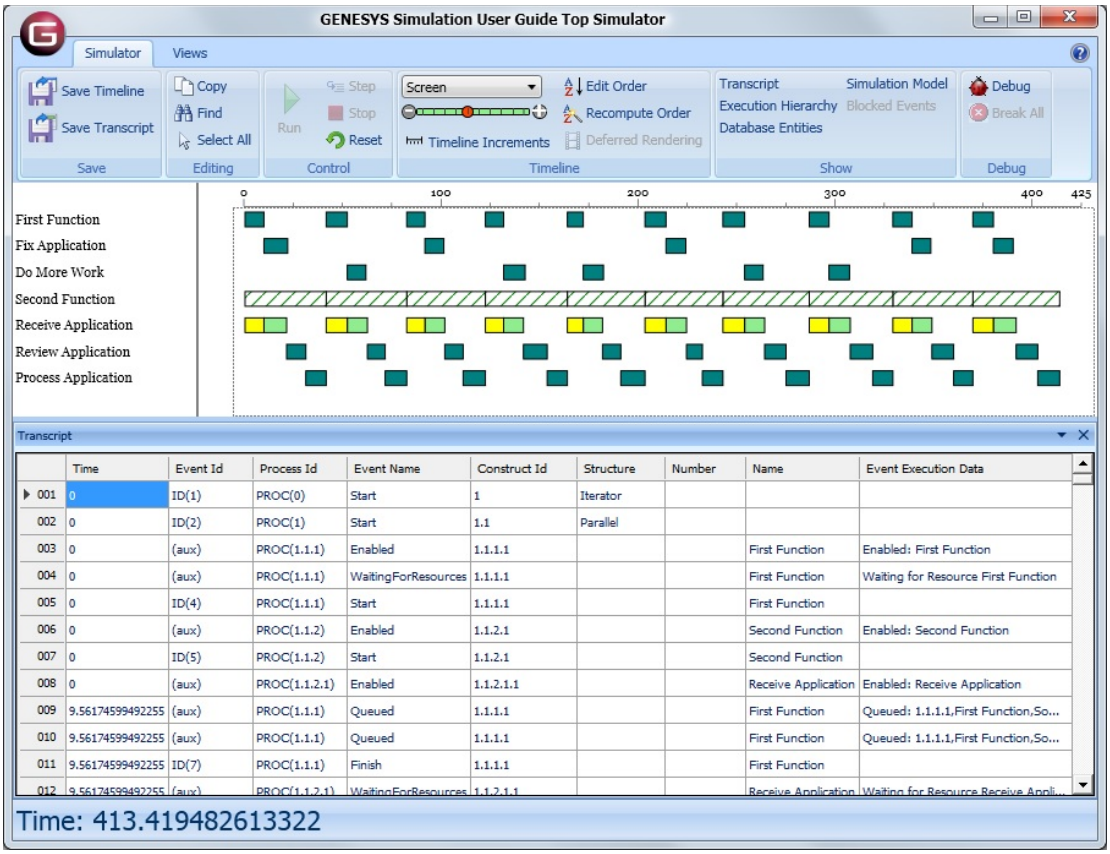


Figure 37 Model Containing Decomposition

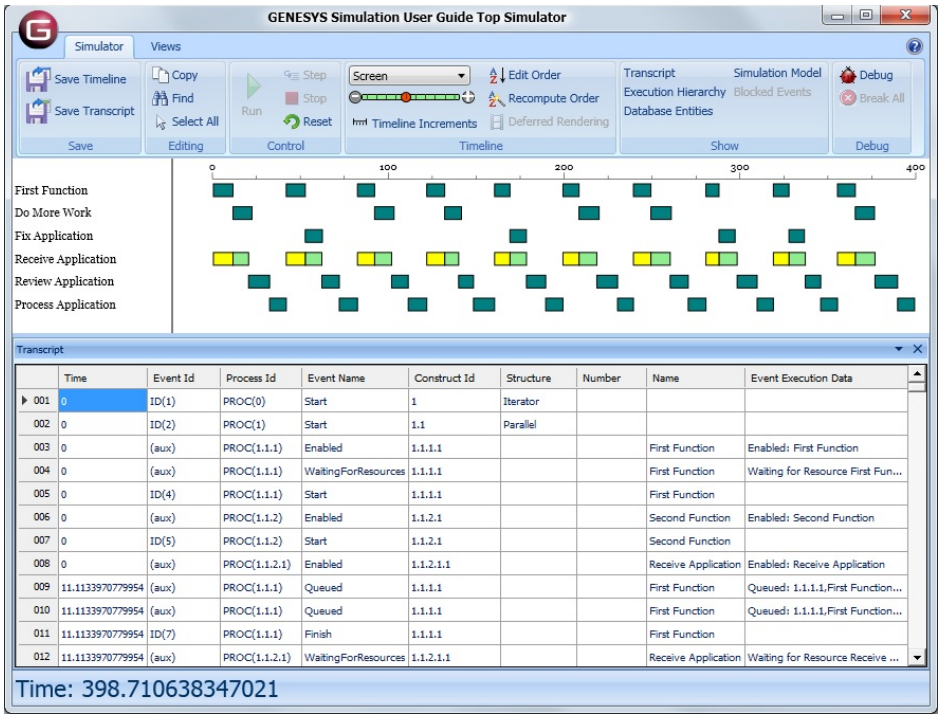


	Time	Event Id	Process Id	Event Name	Construct Id	Structure	Number	Name	Event Execution Data
001	0	ID(1)	PROC(0)	Start	1	Iterator			
002	0	ID(2)	PROC(1)	Start	1.1	Parallel			
003	0	(aux)	PROC(1.1.1)	Enabled	1.1.1.1			First Function	Enabled: First Function
004	0	(aux)	PROC(1.1.1)	WaitingForResources	1.1.1.1			First Function	Waiting for Resource First Function
005	0	ID(4)	PROC(1.1.1)	Start	1.1.1.1			First Function	
006	0	(aux)	PROC(1.1.2)	Enabled	1.1.2.1			Second Function	Enabled: Second Function
007	0	ID(5)	PROC(1.1.2)	Start	1.1.2.1			Second Function	
008	0	(aux)	PROC(1.1.2.1)	Enabled	1.1.2.1.1			Receive Application	Enabled: Receive Application
009	9.56174599492255	(aux)	PROC(1.1.1)	Queued	1.1.1.1			First Function	Queued: 1.1.1.1,First Function,So...
010	9.56174599492255	(aux)	PROC(1.1.1)	Queued	1.1.1.1			First Function	Queued: 1.1.1.1,First Function,So...
011	9.56174599492255	ID(7)	PROC(1.1.1)	Finish	1.1.1.1			First Function	
012	9.56174599492255	(aux)	PROC(1.1.2.1)	WaitingForResources	1.1.2.1.1			Receive Application	Waiting for Resource Receive Appli...

Figure 38 Results of Model Containing Decomposition

There may be times when you do not want to execute the decomposition of a **Function**. If the Execute Decomposition attribute value is set to *false*, then the **Function** rather than its decomposition is executed. By default, the Execute Decomposition is set to *true.* Note the timeline bar with the diagonal lines (Figure 38). This bar represents the duration of the decomposed **Function**, Second Function. The starting time for this **Function** corresponds to the start time of the first subordinate **Function**, Receive Application, and the completion time corresponds to the completion of the last subordinate **Function**, Process Application.

With the Execute Decomposition attribute value of Second Function set to *false*, execution of our example results in the Timeline shown in Figure 39.



	Time	Event Id	Process Id	Event Name	Construct Id	Structure	Number	Name	Event Execution Data
001	0	ID(1)	PROC(0)	Start	1	Iterator			
002	0	ID(2)	PROC(1)	Start	1.1	Parallel			
003	0	(aux)	PROC(1.1.1)	Enabled	1.1.1.1			First Function	Enabled: First Function
004	0	(aux)	PROC(1.1.1)	WaitingForResources	1.1.1.1			First Function	Waiting for Resource First Fun...
005	0	ID(4)	PROC(1.1.1)	Start	1.1.1.1			First Function	
006	0	(aux)	PROC(1.1.2)	Enabled	1.1.2.1			Second Function	Enabled: Second Function
007	0	ID(5)	PROC(1.1.2)	Start	1.1.2.1			Second Function	
008	0	(aux)	PROC(1.1.2.1)	Enabled	1.1.2.1.1			Receive Application	Enabled: Receive Application
009	11.1133970779954	(aux)	PROC(1.1.1)	Queued	1.1.1.1			First Function	Queued: 1.1.1.1,First Function...
010	11.1133970779954	(aux)	PROC(1.1.1)	Queued	1.1.1.1			First Function	Queued: 1.1.1.1,First Function...
011	11.1133970779954	ID(7)	PROC(1.1.1)	Finish	1.1.1.1			First Function	
012	11.1133970779954	(aux)	PROC(1.1.2.1)	WaitingForResources	1.1.2.1.1			Receive Application	Waiting for Resource Receive ...

Figure 39 Execute Decomposition Set to False

4.7 Loop

A Loop construct in a model causes the simulator to repeat the enclosed branch logic. If we replace the Iterate with a loop, the enclosed logic would be repeated indefinitely rather than a fixed number of times. The model in Figure 40 will repeat First Function indefinitely. One would have to click ***Stop*** on the simulator to get the simulation to cease.

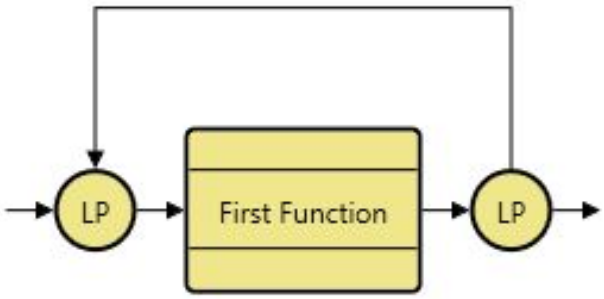


Figure 40 Loop Construct

4.8 Loop Exit

A Loop Exit provides the mechanism by which a loop can be terminated. When the simulator encounters a Loop Exit, the simulator will jump to the innermost closing loop node and execute the **Function** or construct immediately after the loop close. In Figure 41 we have replaced the Iterate with a Loop and added a Loop Exit to the Bad Application branch. We have also changed the selection probabilities.

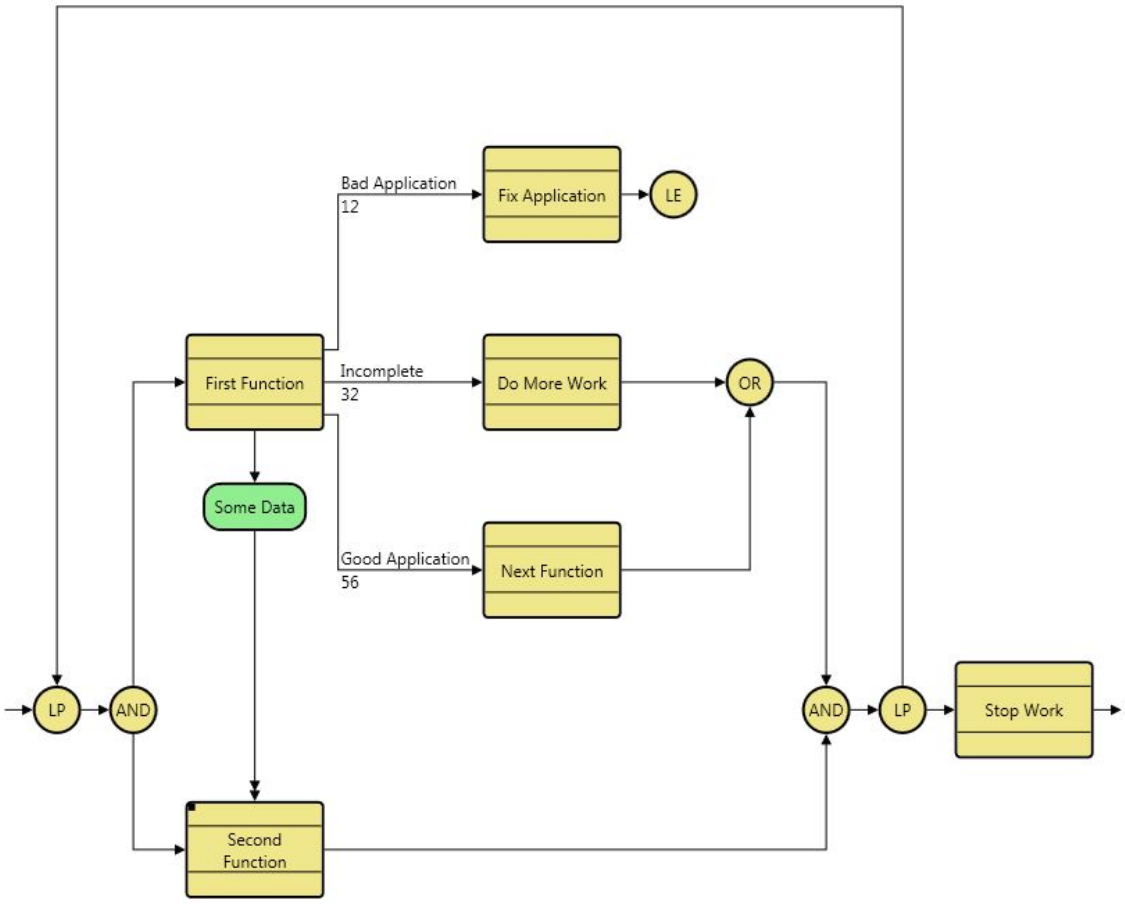
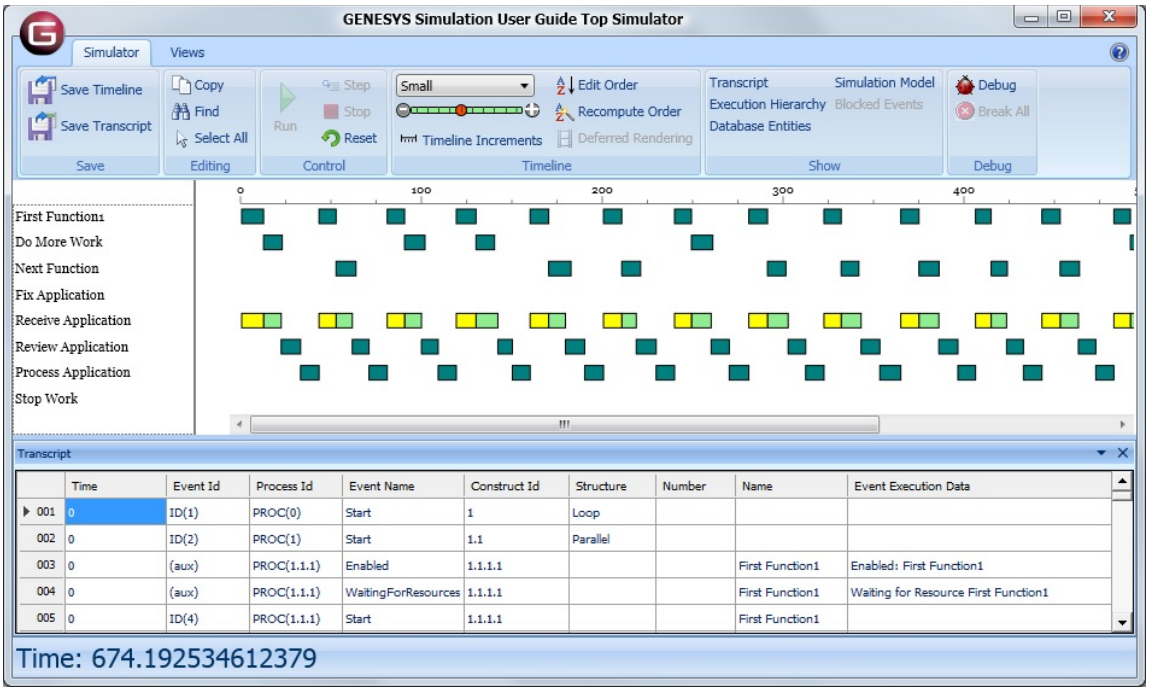


Figure 41 EFFBD with Loop and Loop Exit

4.8.1 Executing a Model with a Loop and Loop Exit

You can see from the timeline below in Figure 42, we looped 6 times before First Function *exits by* the Bad Application exit. After exiting the loop, the next **Function** in sequence, Stop Work, executes to complete the simulation run. If we reset and run again, our results will vary, but we will always stop only after we encounter the loop exit on the Bad Application branch.



	Time	Event Id	Process Id	Event Name	Construct Id	Structure	Number	Name	Event Execution Data
001	0	ID(1)	PROC(0)	Start	1	Loop			
002	0	ID(2)	PROC(1)	Start	1.1	Parallel			
003	0	(aux)	PROC(1.1.1)	Enabled	1.1.1.1			First Function1	Enabled: First Function1
004	0	(aux)	PROC(1.1.1)	WaitingForResources	1.1.1.1			First Function1	Waiting for Resource First Function1
005	0	ID(4)	PROC(1.1.1)	Start	1.1.1.1			First Function1	

Figure 42 Results for Model with Loop and Loop Exit

4.9 Replicate

Sometimes multiple instances of identical sets of functions and logic need to operate simultaneously. We designate this by enclosing the replicated logic with a replicate construct.

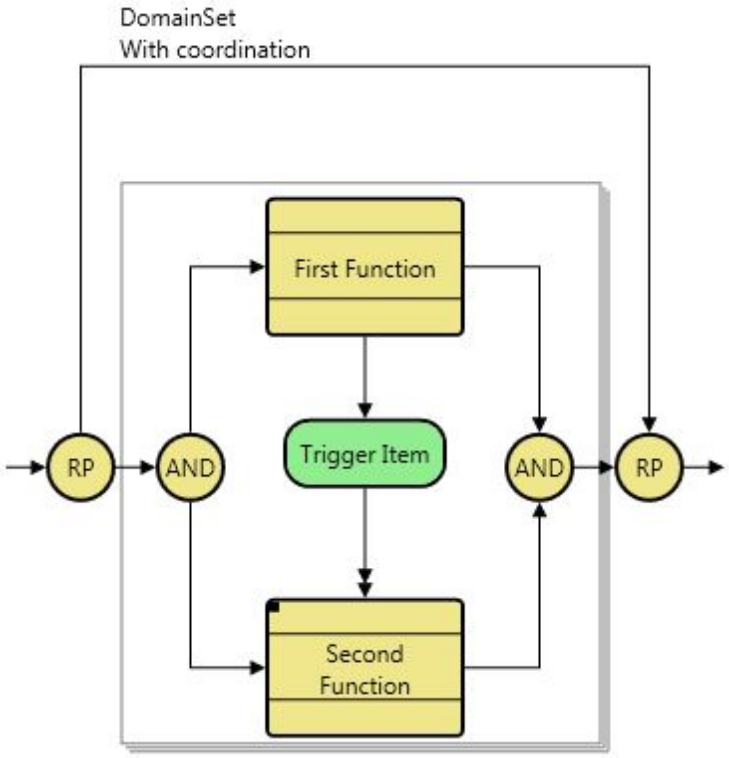
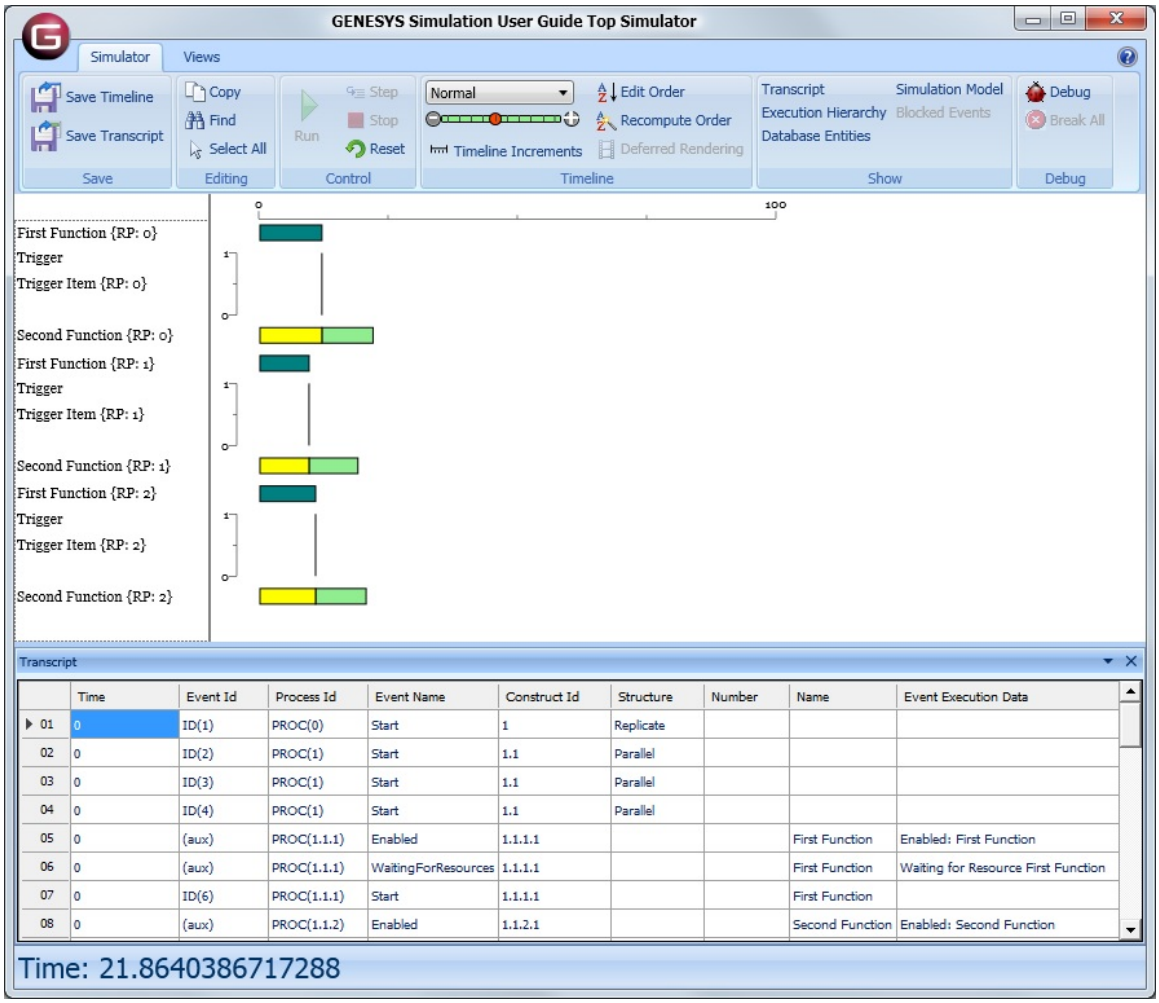


Figure 43 A Replicate Construct

When the simulator is executed on the model shown in Figure 44, the results (Figure 45) show that the three instances of the replicated logic are executed simultaneously. The number of replicated instances is specified by either the system default setting or by the DomainSet defining the replicate. With each replicate, instantiated Functions are enabled at the same time, with each instance of First Function executing immediately and Second Function waiting for the trigger Item from its associated First Function before it is able to execute.



	Time	Event Id	Process Id	Event Name	Construct Id	Structure	Number	Name	Event Execution Data
01	0	ID(1)	PROC(0)	Start	1	Replicate			
02	0	ID(2)	PROC(1)	Start	1.1	Parallel			
03	0	ID(3)	PROC(1)	Start	1.1	Parallel			
04	0	ID(4)	PROC(1)	Start	1.1	Parallel			
05	0	(aux)	PROC(1.1.1)	Enabled	1.1.1.1			First Function	Enabled: First Function
06	0	(aux)	PROC(1.1.1)	WaitingForResources	1.1.1.1			First Function	Waiting for Resource First Function
07	0	ID(6)	PROC(1.1.1)	Start	1.1.1.1			First Function	
08	0	(aux)	PROC(1.1.2)	Enabled	1.1.2.1			Second Function	Enabled: Second Function

Figure 44 Results for Model with A Replicate Construct

4.10 Kill Branch

When a kill branch is completed, all functions within the parallel construct immediately stop and control is transferred to the construct following the parallel construct. This kill status specification permits you to model termination paths for concurrent behavior. If specified as a kill branch, the label "kill" will be displayed at the front of the branch below the control flow line.

As shown below in Figure 45, the model consists of three branches; two of these branches contain loops without exits. By designating the third branch as a kill branch, the loop branches will terminate upon execution of the third branch. Often Parallel construct branches will contain infinite Loop constructs that need to be terminated simultaneously upon a specific event. Adding an additional parallel branch and designating the branch as a “kill“ branch, as illustrated below, represents this.

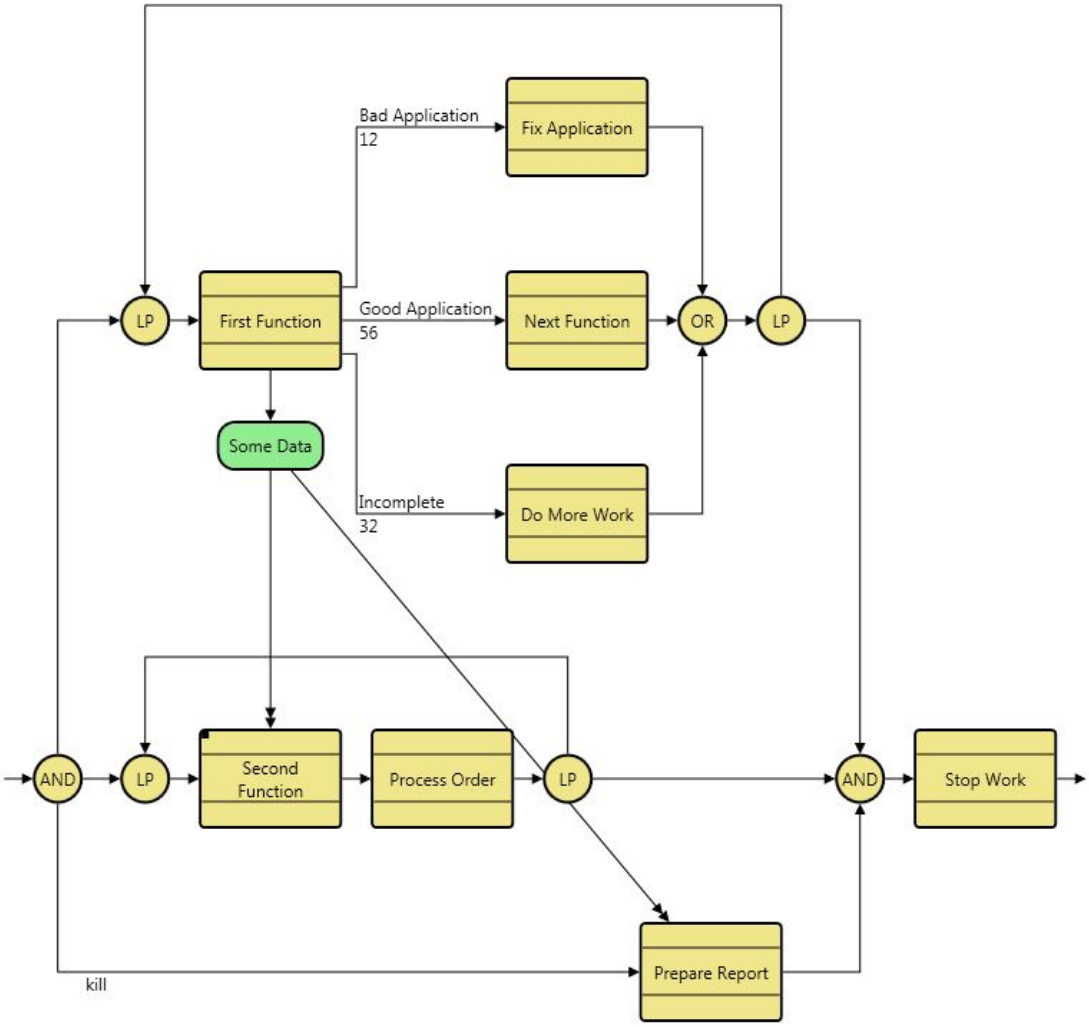


Figure 45 Model with Kill Branch

When a kill branch completes its execution, all executing **Functions** within the parallel construct immediately stop and control is transferred to the construct following the parallel rejoin, in this case, the **Function** Stop Work. Any **Resources** captured by these **Functions** will be released; however, these **Functions** will neither *produce* any **Resources** nor generate any **Items**. In our model, we exited the parallel construct when Prepare Report completes execution. The results are shown in Figure 46.

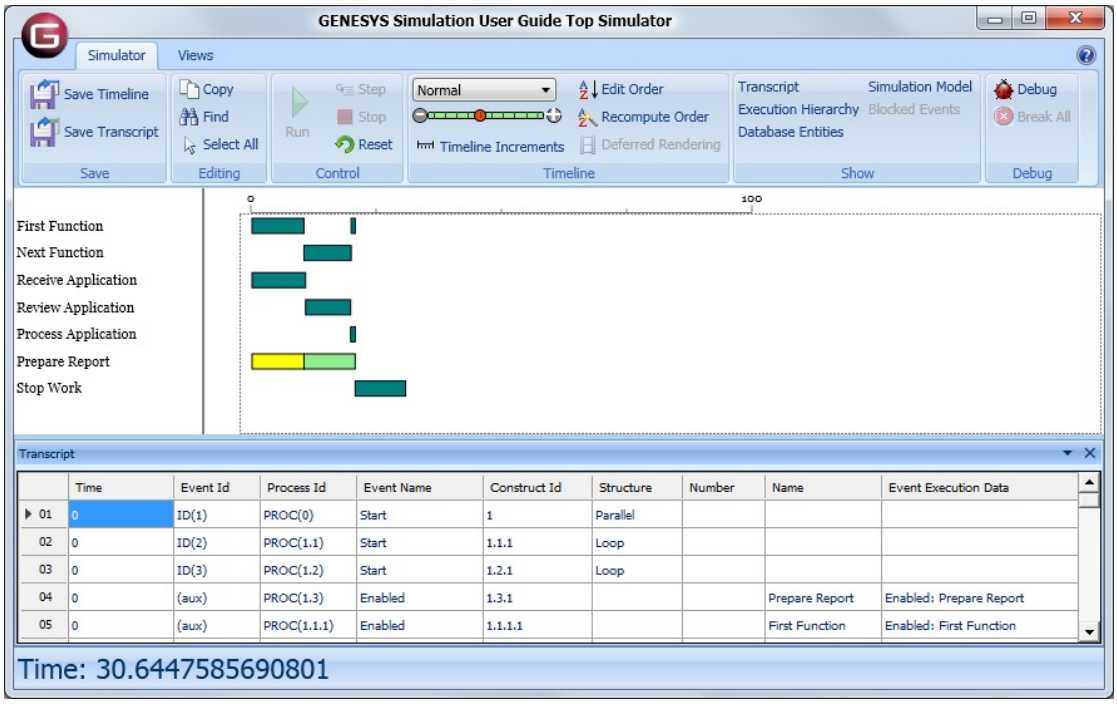


Figure 46 Simulation Results with Kill Branch

4.11 Executing a Model with Links and Items

GENESYS provides a modeling capability that integrates both the functional and physical views of the system. The modeling semantics are shown in the schema diagram below (Figure 47). The simulator is a valuable tool for evaluating the feasibility of a proposed architecture once the functional allocations and the association of the **Items** to the **Links** is completed. The systems engineer will be able to evaluate the architecture's performance and possibly discover unexpected results. The following examples show how **Link** Delays can affect overall system timeline.

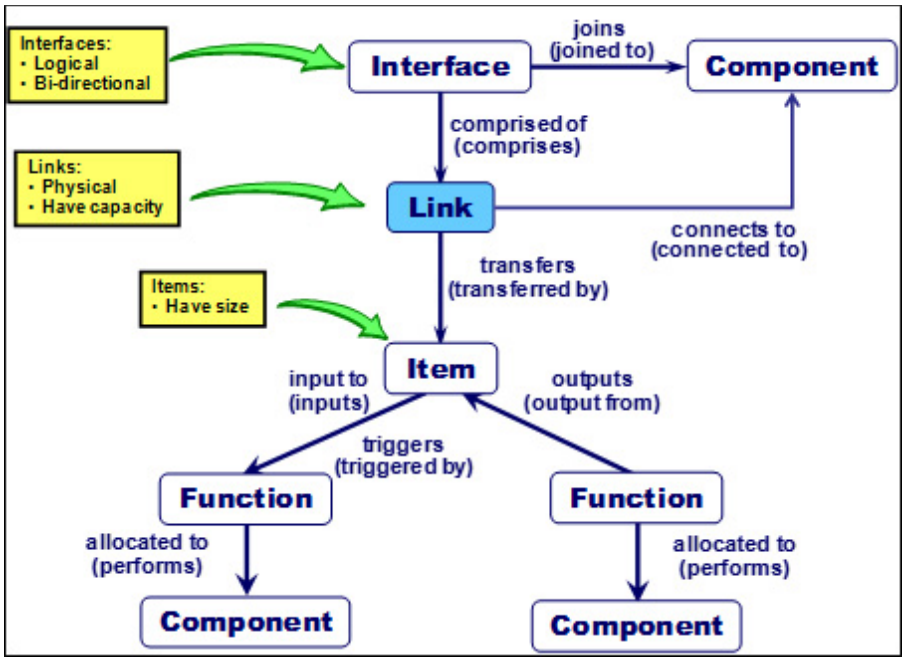


Figure 47 Interface and Link Schema Diagram

The GENESYS simulator evaluates three attributes (Delay, Size and Capacity) within the **Item** and **Link** classes respectively when it computes the total time for a **Link** to transfer an **Item** as shown in the following equation:

Total **Link** Time = [**Link** Delay + (**Item** Size/**Link** Capacity)]

To show the impact on performance, a model was created and run with variation of these attributes. This model is shown in Figure 48. Note Some Data is *transferred by* a **Link**, Transfer Link.

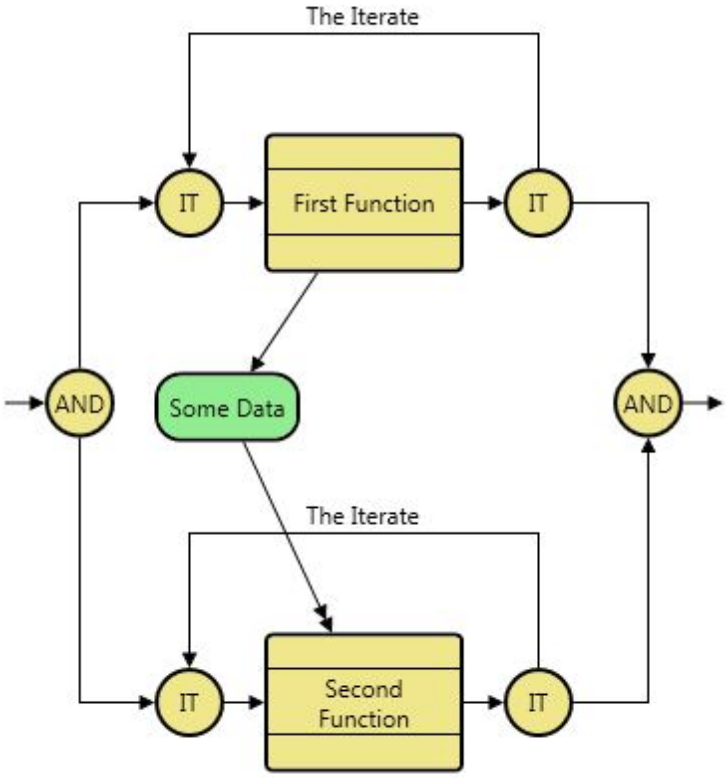


Figure 48 Link-Item Model

Initially the **Item** Size is set to 0.0; the **Link** Capacity is set to 1000.0 with a Delay of 0.0. Simulation results with these values are shown in Figure 49. Note there is no delay. Once First Function completes execution, the trigger, Some Data, is created and triggers Second Function, allowing the start of execution.

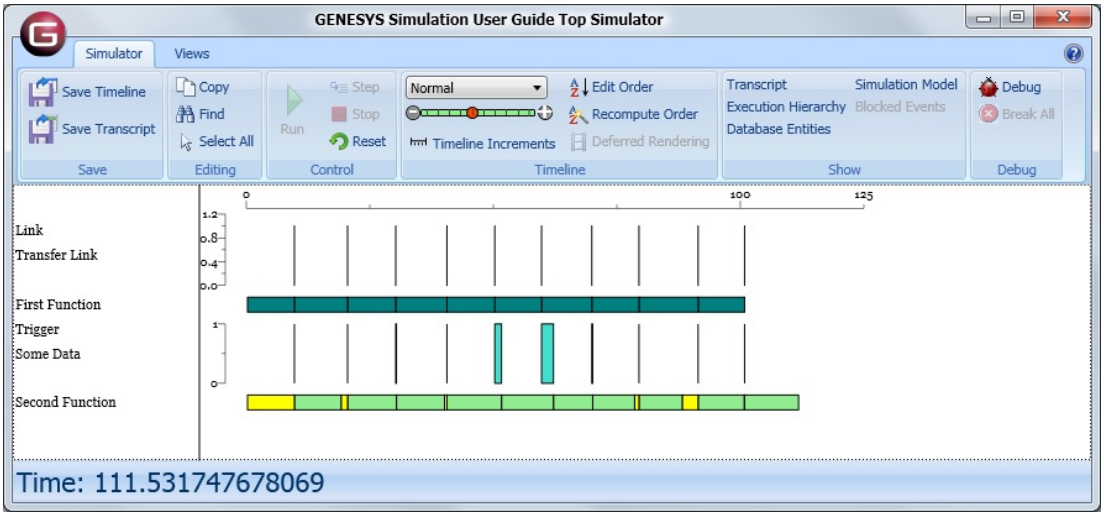


Figure 49 Simulation Results with Item Size = 0.0, Link Capacity = 1000.0, Delay = 0.0

For the second simulation run, the Item size is increased to 1000.0. Results for this simulation run are shown in Figure 50. Note the total run time is increased to 117.27 due to the delay caused by the time necessary for the **Link** to transfer the item.

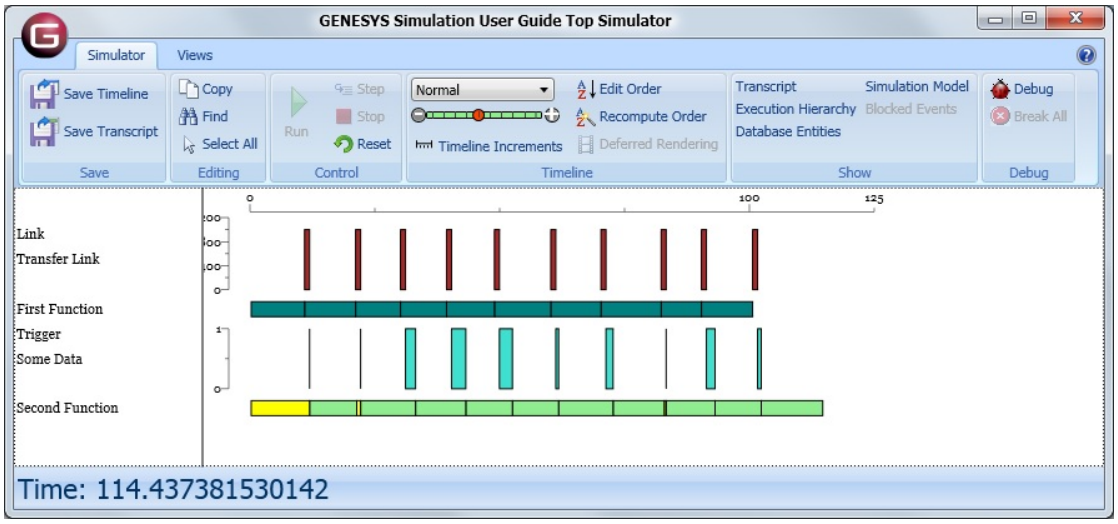


Figure 50 Simulation Results with Item Size = 1000.0, Link Capacity = 1000.0, Delay = 0.0

Lastly, as shown in Figure 51, the **Link** Delay is increased from 0.0 to 0.5. The overall system performance is further degraded due to this **Link** Delay.

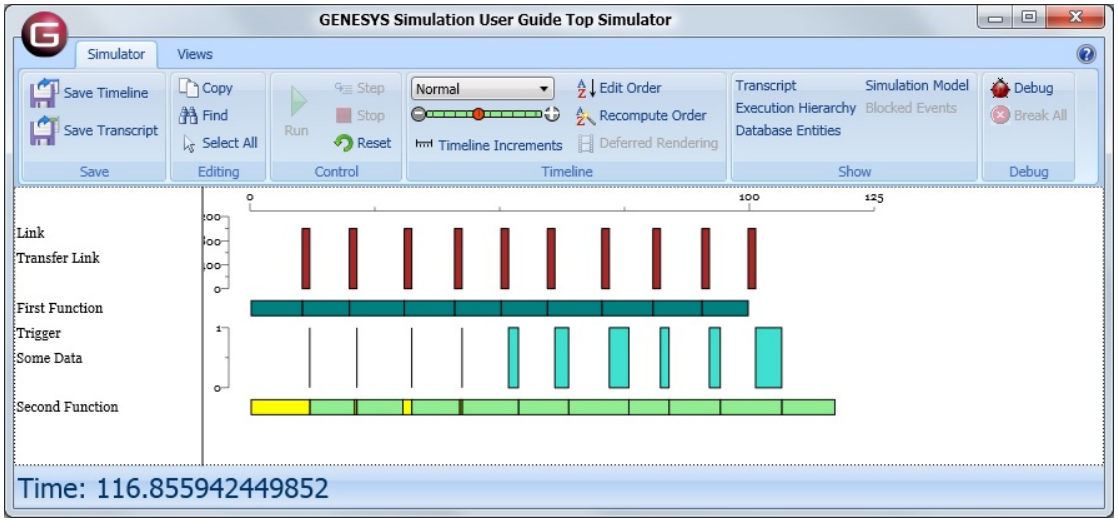


Figure 51 Simulation Results with Item Size = 1000.0, Link Capacity = 1000.0, Delay = 0.5

In summary, GENESYS provides the means to define interfaces between Components, and with the GENESYS simulator, the functional interfaces of the system can be evaluated to determine the effects of interfaces on the system design.

4.12 Execution Order

A simulation run is defined as an execution path through the functional model. The execution of a **Function** is constrained by its Duration, Exit Logic, and any specified Timeout as well as **Resource** availability. **Resource** availability during a simulation run is dependent upon the Initial Amount of the **Resource**, the Maximum Amount of the **Resource** available and the rate of production and consumption of the **Resource**. As discussed later, scripts provide a mechanism to dynamically control these factors affecting execution. Therefore, the order in which scripts are executed is significant and must be considered when building a simulation model using scripts. The following discussion assumes these attributes capitalize on the use of scripts and provide a discussion on the order of execution.

When the GENESYS simulator is opened or the reset command is issued, the simulation model is initialized. The initialization process evaluates the Initial Amount and Maximum Amount of all **Resources** and traverses the functional flow structure of the top level function on which the simulator was opened. Each branch encountered in the traversal constructs is traversed from left to right. Children branches of constructs are traversed from top to bottom. When a **Function** construct is encountered, its decomposition is traversed if the Execute Decomposition attribute value of the **Function** is set to *true*.

As each Function construct is encountered during the initialization process, **Resources** and **Links** associated with the **Function** are initialized. A **Resource** is associated with a **Function** if the **Function** *captures*, *consumes*, or *produces* the **Resource**. A **Link** is associated with a **Function** if the **Function** *inputs*, *outputs*, or is *triggered by* an **Item** carried by the **Link**. **Resources** are initialized before links; however, no particular order is followed in the initialization of the **Resources** or the initialization of the **Links**. For example, if a **Function** outputs multiple **Items** carried by **Links**, the **Links** are not processed in any particular order.

4.13 Resource execution order

When a **Resource** is encountered during the model execution, the Maximum Amount attribute is evaluated followed by the Initial Amount attribute.

4.13.1 Link execution order

With respect to the execution model, **Links** are defined by their Capacity and Delay. When a **Link** is encountered, the **Link** Capacity is evaluated followed by the evaluation of the **Link** Delay.

4.13.2 Function and ProgramActivity Script Execution Order

When the simulation model is executed, the functional flow structure is traversed in the same order as during the initialization process. Most of the behavior exhibited by the simulation model is defined by the Processing Units (elements of the **Function** and **ProgramActivity** class) in the domain model.

When the flow of control moves to a function and the function is enabled, attributes are evaluated in the following order:

1. The Begin Logic attribute is evaluated. The Begin Logic is where any preconditions on the function are set before the normal function behaviors are executed.
2. The Timeout attribute is evaluated.
3. The *captures* Amount attribute is evaluated for each *captures* relationship. There is no specific order when there are multiple *captures* relationships for the function.
4. The *consumes* Amount attribute is evaluated for each *consumes* relationship. There is no specific order when there are multiple *consumes* relationships for the function.

After all triggers (**Items**) and **Resources** required by a **Function** are available, the **Function's** Duration attribute is evaluated to determine the completion time of the **Function** when the **Function** start event occurs.

When a **Function** is going through the last phase of its execution during the finish event, attributes are evaluated in the following order:

1. **Item** Size attribute is evaluated for each *outputs* relationship. There is no specific order when there are multiple *outputs* relationships for the **Function**.
2. *Produces* Amount attribute is evaluated for each *produces* relationship. There is no specific order when there are multiple *produces* relationships for the function.
3. If the **Function** is a multi-exit function, the Exit Logic attribute is evaluated.
4. The End Logic attribute is evaluated. The End Logic script is where any post conditions can be performed.

However, if the Functions are located in a branch other than the kill branch and the kill branch completes execution before the completion of the Function, the Function will terminate execution at the same time that the kill branch completes execution. If this termination time is before its normal execution, i.e. at the end of the Function's duration, then

1. The Items will not be generated.
2. Resources will not be produced.
3. The Exit Logic attribute will not be evaluated.
4. The End Logic attribute will not be evaluated.

DomainSet Script Execution Order

The Count attribute of the **DomainSet** on an Iterate construct determines the number of iterations through the construct. The **DomainSet** Count attribute is evaluated on the first iteration through the branch when flow of control moves to an iterate.

THIS PAGE INTENTIONALLY BLANK

5 ADDING RESOURCES

Adding resources to your model allows users to investigate their systems' resource requirements and effects of resource contention on the timeline. The GENESYS base schema contains the **Resource** class. **Resources** can be *produced by*, *consumed by*, or *captured by* a **Function** when running the GENESYS simulator. The simulator will account for the production, consumption, or usage of these resources during execution of the model. As previously discussed in Section 1.6, the resources may be displayed on the simulator timeline.

Resources are *consumed* when a **Function** begins its execution, and are *produced* when the **Function** terminates its execution. If a **Function** *captures* a **Resource**, the **Function** acquires the **Resource** when it starts execution and releases the **Resource** when it terminates. Refer to Section 4.12 for more information on order of execution.

Captured resources are not consumable; rather, they represent renewable resources such as computer memory. A renewable resource is captured (and held) at the beginning of the Function's execution and released at the end of the Function's execution.

Consume/Produce In Lieu of Capture
If all functions on a branch *capture* a **Resource**; instead of having each **Function** *capture* the **Resource**, have the first **Function** on the branch *consume* the **Resource** and have the last **Function** on the branch *produce* the **Resource**.

The amount of **Resource** *consumed*, *produced*, or *captured* is designated by the Amount attribute of the corresponding relationship. The Amount attribute can be specified as a constant, random number, or the result of executing a script. This uses the NumberSpec described in Section 2.4.

The Acquire Available attribute on these relationships is a Boolean value (*true*/*false*) that specifies whether the **Function** should reserve the available resources while waiting for the full amount to be available. If set to *false*, the **Function** waits for the full amount to become available before it acquires the **Resource** Amount it needs to execute. If set to *true*, the **Function** acquires the amount of **Resource** currently available and waits for the remainder of the **Resource** to become available before it executes.

5.1 Executing a Model with Resources

Using our simple example, EFFBD with Loop and Loop Exit (Figure 41), we have modified the model by adding a **Resource,** MIPS, and deleting the trigger. When a **Resource** is displayed in the timeline window, the amount of the **Resource** is displayed to show the available amount of the **Resource** at any given time. If a **Resource** is demanded by a **Function**, the **Resource** amount is deducted from the amount available and the timeline display is decremented to show the amount remaining. If the Acquired Available relationship attribute is set to *true* and an insufficient amount is available, then the **Resource** is highlighted in red to indicate that the **Resource** is reserved for a **Function**. In addition, the **Function** that requested the **Resource** will not execute until the full amount is available, indicating a starved **Function** waiting for a **Resource**. The delay-time for the **Function** from requesting the resource amount until acquiring it is depicted on the timeline by a magenta colored timeline bar preceding the execution of the **Function**.

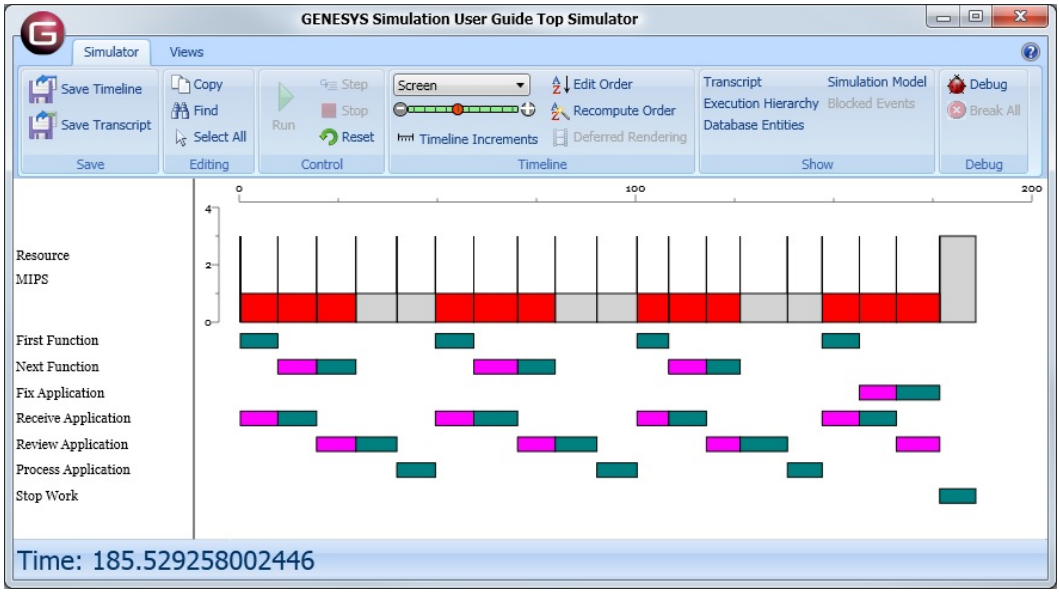


Figure 52 Loop Model with Insufficient Resources

As shown in the timeline above (Figure 52), there are insufficient MIPS, as shown in red, that result in execution delay of three **Functions**, the delays shown as magenta. The obvious solution to improve performance is to increase the **Resource** from an Initial Amount of 3 to an Initial Amount of 6. Upon resetting and running the simulation with increased MIPS, the timeline (Figure 53) shows that the resources are now more than adequate and eliminate any delay in execution. Further analysis can be accomplished to optimize the design such that the least amount of resources is specified for the desired performance.

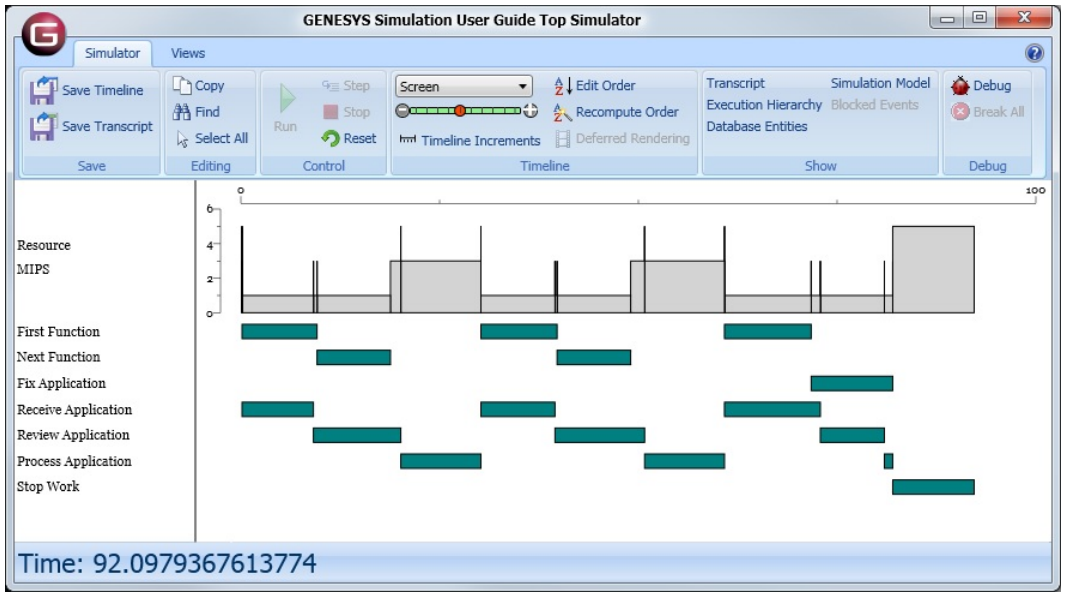


Figure 53 Loop Model with Adequate Resources

Resources can also be added to a model to provide control. Specifically, **Resources** can function as triggers. For example, one **Function** produces a trigger-type **Resource** that is then consumed by a subsequent **Function** awaiting the trigger-type Resource. Again, going back to our Loop Exit model, a Resource, Start Branch has been added to the model. First Function *produces* Start Branch which is *consumed by* Receive Application. The simulation results are shown in Figure 54. Note that Receive Application is delayed (magenta bar) until First Function completes execution, producing the **Resource**, Start Branch.

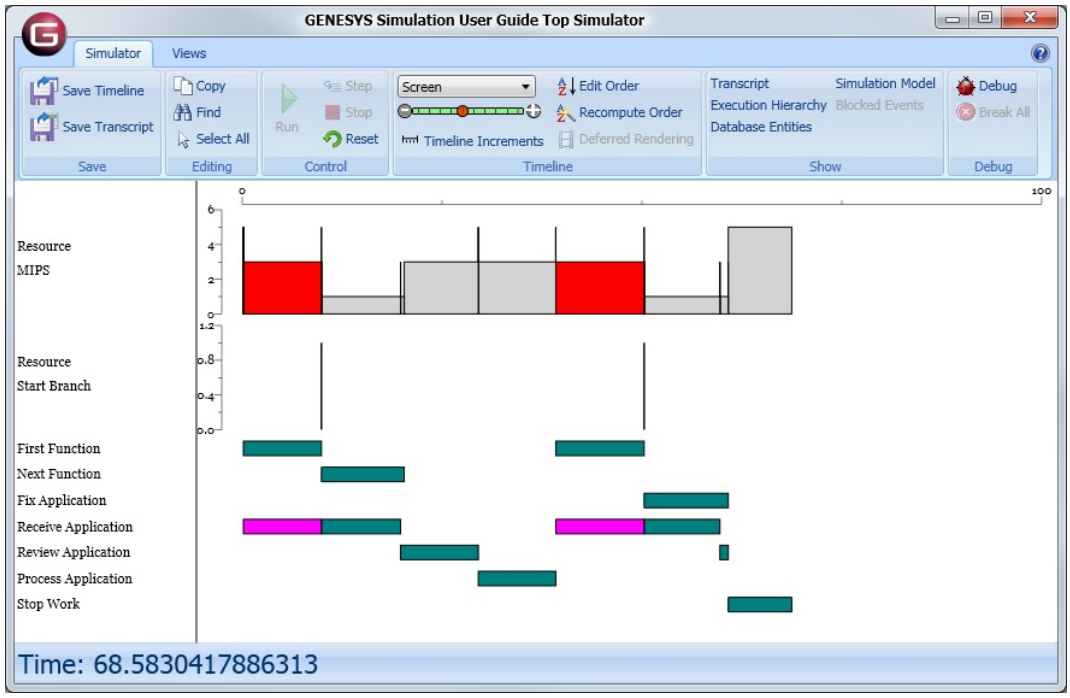


Figure 54 Loop Exit Model with Trigger-Type Resource

THIS PAGE INTENTIONALLY BLANK

6 ENHANCING THE SIMULATOR MODEL USING SCRIPTING

GENESYS uses the Microsoft Visual Basic .NET language to allow users to specify computations and logic to detail the behavior and resource models. The power of the GENESYS simulator model is enhanced by incorporating scripting. The model becomes much more powerful, allowing the user to vary input conditions and logic from one simulation run to the next as well as collecting data in alternative format other than the timeline for detailed analysis. This section is intended to provide a brief introduction to the use of Microsoft Visual Basic within the GENESYS simulator.

6.1 Script Basics

Scripts, written using Microsoft Visual Basic[2], are added to a GENESYS simulator model through the Edit NumberSpec script window or the Edit ScriptSpec Window; an example window is shown in Figure 55. These windows can be accessed by selecting the ***Edit*** button for the applicable attribute or double-clicking the relationship attribute then selecting the script button. This window consists of the following parts: the Editor ribbon at the top of the window; the Script pane on middle left; the Error pane on the lower left; the Solution Browser pane on the middle right; the Explorer pane on the lower right.

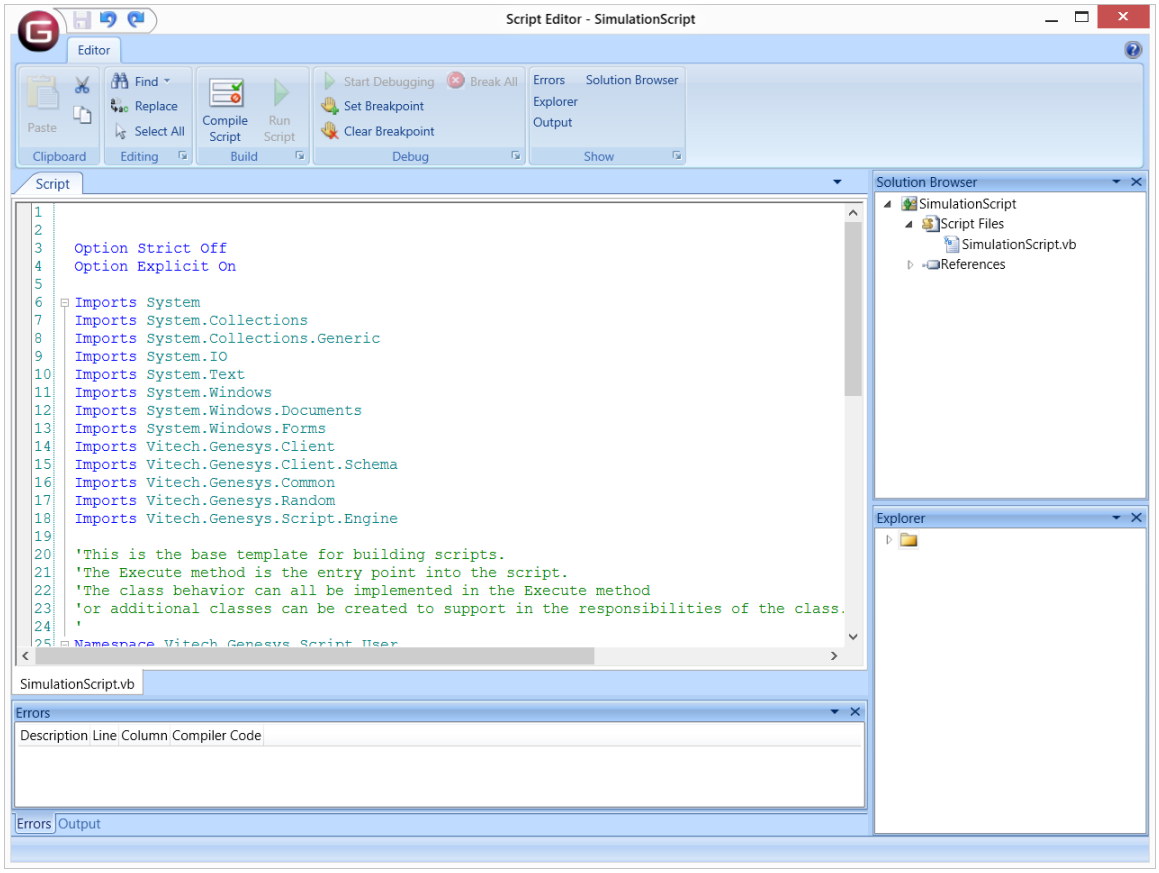


Figure 55 Edit ScriptSpec Window

[2] NOTE: The version of Visual Basic used within GENESYS for scripting is Visual Basic .NET 4.0. GENESYS does not use Visual Basic for Applications nor VBScript.

6.2 GENESYS Application Programmer's Interface

The user's scripts, written in Visual Basic, access the GENESYS repository, and hence the project data, through the GENESYS Application Programmer's Interface (API). Details on using the API are given in the **Getting Started with GENESYS API** document, which is included in the GENESYS documentation.

APPENDIX A - GENESYS SIMULATOR DISTRIBUTIONS

GENESYS offers the advanced simulator user the ability to perform simulations that are more complex. These simulations emulate conditions that are more realistic where events and timing possess random elements. GENESYS uses two primary features for better using the simulation results to effect change to the desired system behavior.

The first feature is the selection and specification of the random distribution characteristics. One may choose from one or several probability distributions from which to draw random numbers, e.g., binomial, Bernoulli, Gaussian, Poisson, etc. Once the probability distribution is choen, values for the parameters that govern the behavior of the probability distribution can be chosen next, such as the *mean* and *variance* for the Gaussian probability distribution.

The second primary feature is the choice of which *Random Number Stream* serves as the random value selection source for any given simulation variable. Multiple Random Number Streams are offered because this capability allows users to use multiple probability distributions in the simulation and it offers a set of uncorrelated simulation variables when drawing random variables from the same probability distribution.

For repeatability, GENESYS allows users to specify which Random Number Stream to begin with. To get this repeatability, GENESYS allows users to specify the beginning Random Number Stream. The stream can be viewed from the ***User Random Streams*** tab of ***Preferences***. Table 11 provides some of the more common parameters that characterize random variable definitions.

Whether setting the default random variable distribution or editing an attribute such as Duration in the class **Function**, clicking the ***Random*** tab or ***Random*** radio button enables random distribution editing. A drop-down list allows users to choose the Distribution type, e.g., binomial, uniform, etc. For the random distribution selected, certain parameters governing the characteristics of the random distribution are available for editing. The following section provides a brief description of the preset random probability distributions, and their governing parameter definitions.

Random Variable n – A variable whose value randomly changes as a function of one or more parameters. GENESYS uses discrete random variables. Discrete random variables take their values from a finite ordered collection in which the members can be placed in any order.

Random Number Stream – In GENESYS, a sequence of random numbers is captured and associated with a random number stream. Each random number stream is "randomly" generated using an appropriate random number generator with using some set of initial conditions. One may use the same set of initial conditions to generate the same sequence of random variables, or users may want to begin each simulation with a specific "random" value.

Table 11 Random Variable Definitions

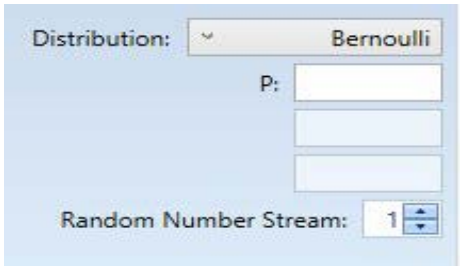
Parameter	Definition
Mean	Weighted average or expected value; provides a measure of the most likely value of the distribution
Variance	Provides a measure of the spread of the random variables relative to the mean of the distribution.
PDF	Probability density function. The PDF is used to determine the probability that a random variable will lie within a range of values.
Random number stream	A sequence of random numbers

A brief description of the preset random probability distributions and their governing parameter definitions is provided below. The graphical representation and details about of these distributions is taken from Discrete Event Simulation in C by Kevin Watkins.

There is much more to random number methodology and the concepts of probability than what is presented here. For additional study, in addition to Discrete Event Simulation in C by Kevin Watkins, we recommend Systems Engineering and Analysis by Benjamin S. Blanchard and Wolter J. Fabrycky.

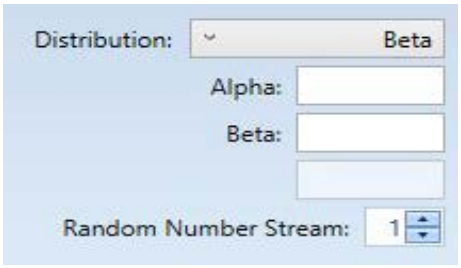
Bernoulli

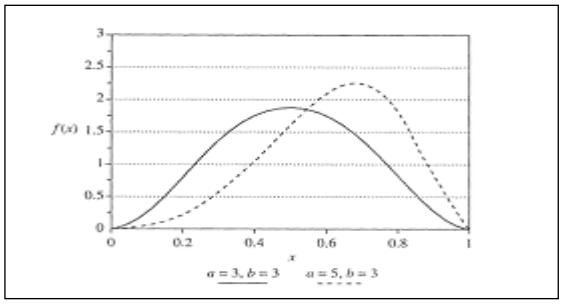
The Bernoulli distribution specifies the probability of a successful event as “p.” and the probability of failure of that event as “1-p.” Given “p,” the Bernoulli distribution calculates the value to have a successful outcome (such as a time duration of one) if the random number is “<= p.” Otherwise, the distribution calculates the value to have an unsuccessful outcome (such as a time duration of 0) if the random number is “>p.” This returns a Boolean value (True or False) that determines if the event was successful. If used to specify a function duration, the value of p must be > 0 and <= 1.0, and if the result is True, then the Function executes with a duration of 1. If False, the Function executes with a duration of zero.



Beta

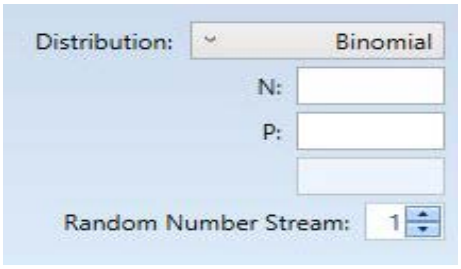
The Beta distribution is useful when you need to generate a continuous random variable between fixed bounds. For instance, use it to model random proportions such as the fraction of packets requiring re-transmission in a data transmission link. The beta distribution has two parameters, alpha, which affects the degree of skew, and beta, which affects the peak, where alpha > zero and beta > zero.

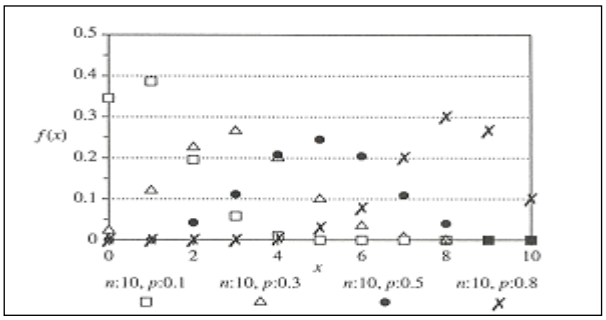




Binomial

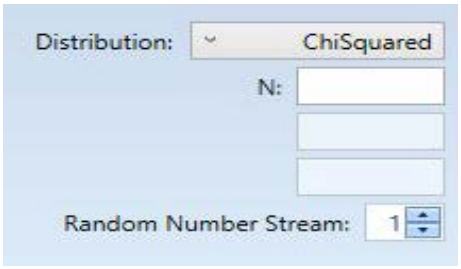
The Binomial distribution is used to model the number of successes in a sequence of *n* independent trials.

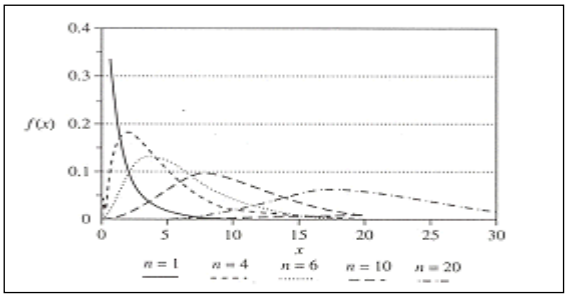




Chi-Squared

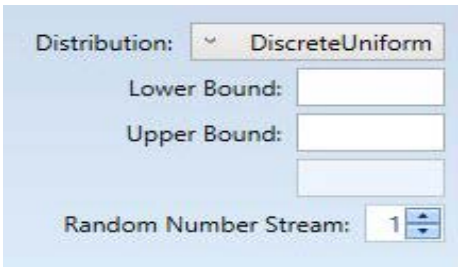
The Chi-Squared distribution with n degrees of freedom is the sum of *n* independent standard normal deviates squared.

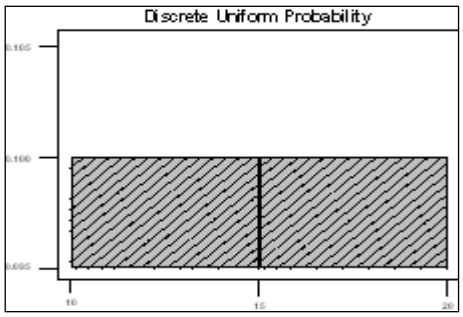




Discrete Uniform

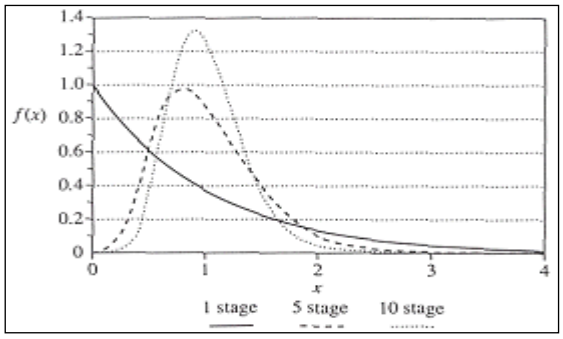
In the Discrete Uniform distribution, each value of the random variable is assigned identical probabilities.





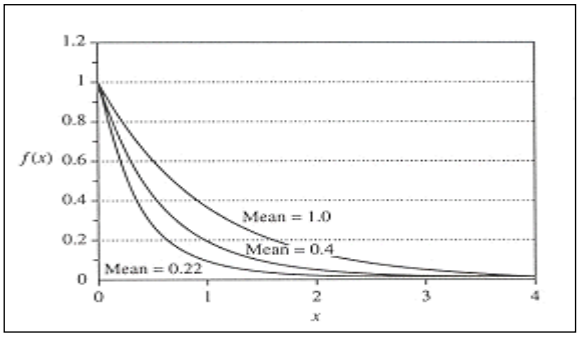
Erlang

The Erlang distribution of order *r* is the waiting time to the r^{th} event in a so-called Poisson process. It is used to model service times.



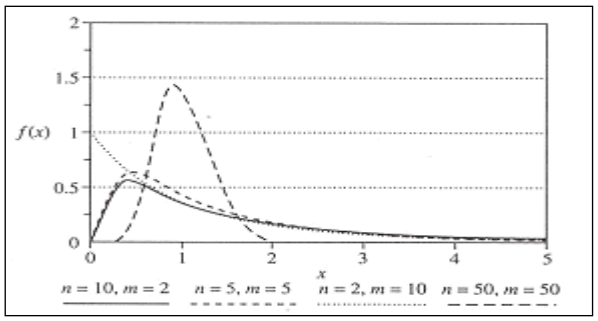
Exponential

The Exponential distribution is used to model purely random events such as the time between failures or the time between arrivals of a customer.



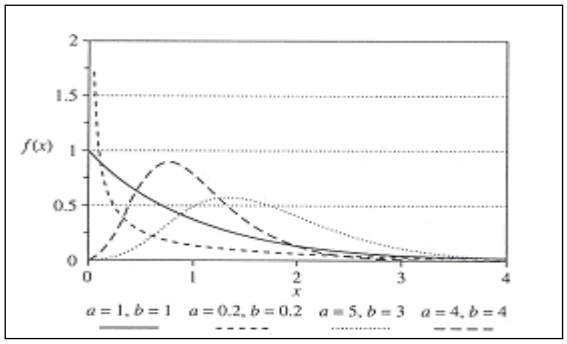
F

The F distribution is often used for testing variances.



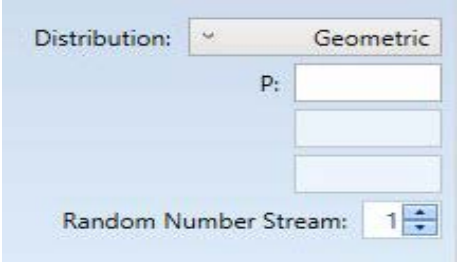
Gamma

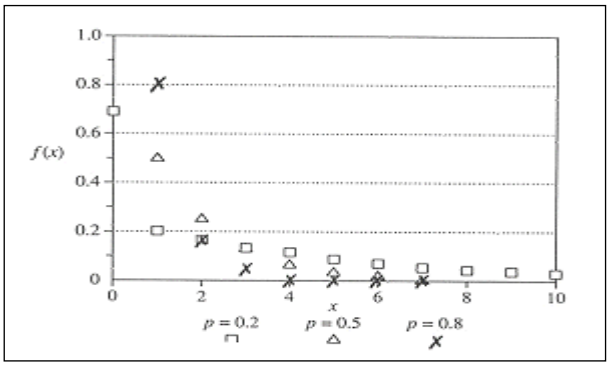
The Gamma distribution is a commonly used distribution that is also suitable as the basis for the generation of random variables for several other distributions.



Geometric

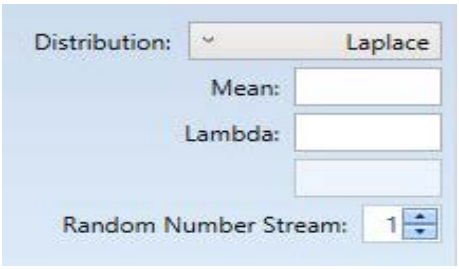
The Geometric distribution represents the number of trials that occur in a sequence of Bernoulli trials until the first success is encountered.

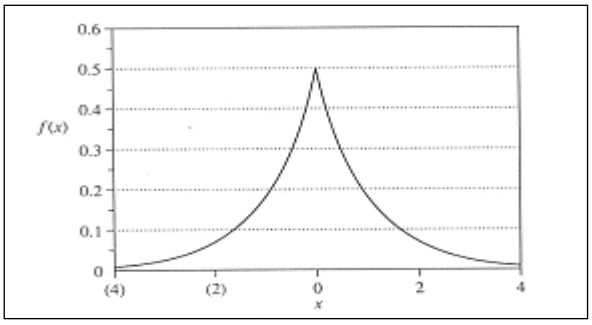




Laplace

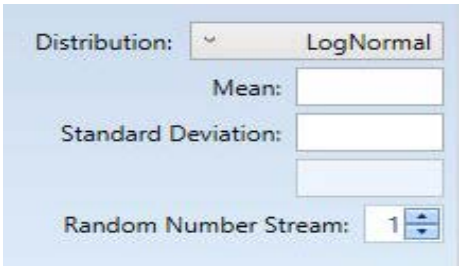
The Laplace distribution looks like an exponential distribution with a mirror image in the y-axis.

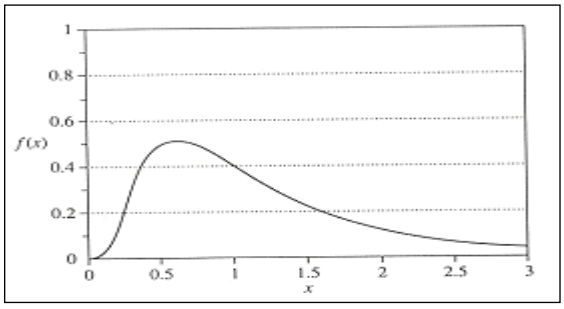




Lognormal

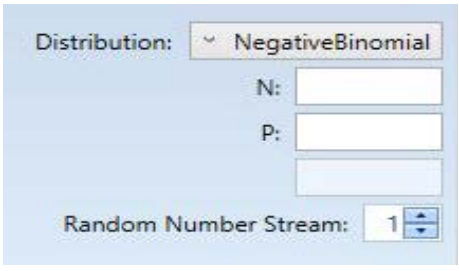
The product of a large number of positive random variables tends to have a lognormal distribution.

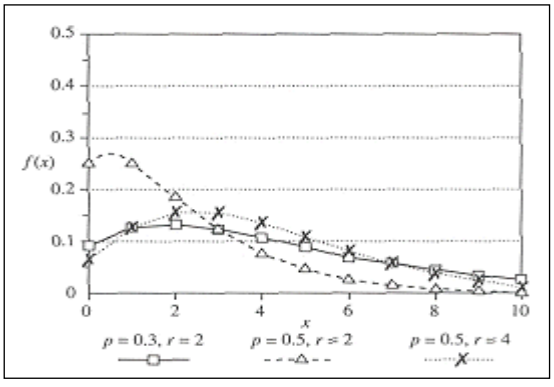




Negative Binomial

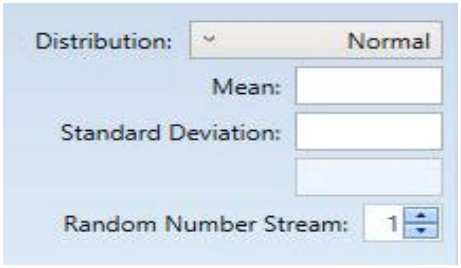
The negative binomial distribution gives the number of failures before the n^{th} success.

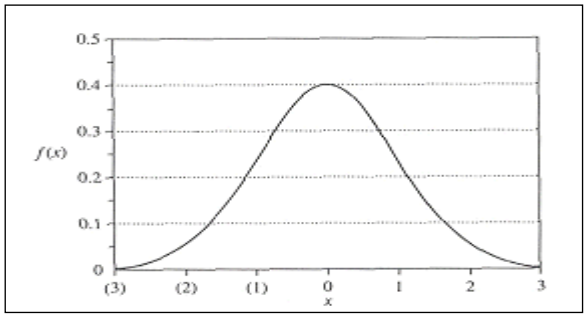




Normal

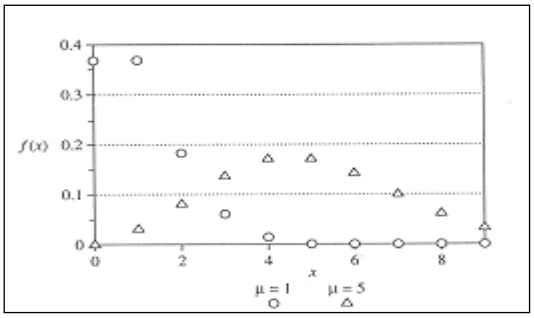
The Normal or Gaussian probability distribution is used extensively because it is useful for describing many statistical processes.





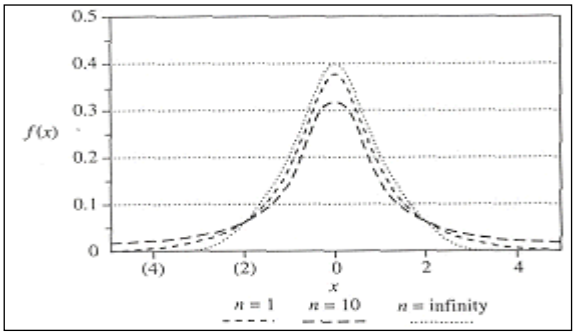
Poisson

The Poisson distribution is the limiting case of the Binomial distribution. This distribution is useful when the opportunity for the occurrence of an event is large, but when the actual occurrence is unlikely. The mean and variance of this distribution are equal, thus there is only one value to enter.



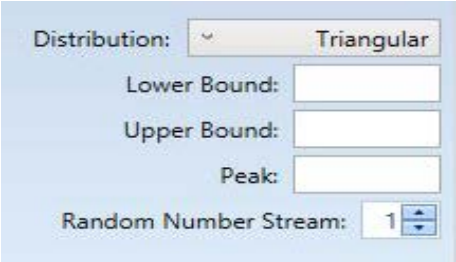
T (Student's T)

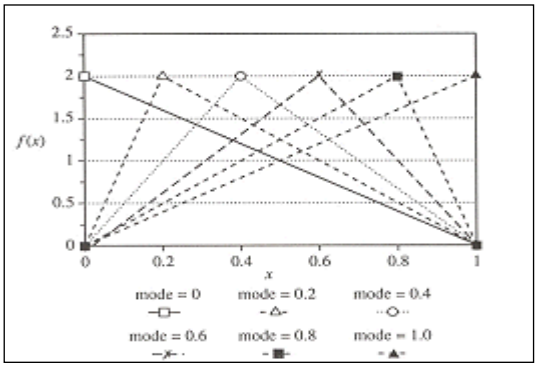
The T distribution is often used for defining confidence intervals.



Triangular

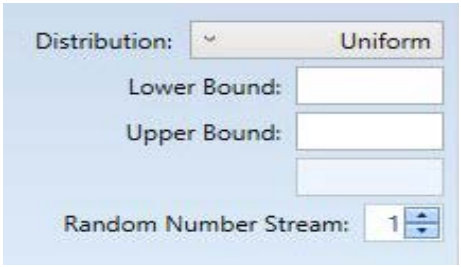
The triangular distribution is a simple way to obtain random variables whose distribution functions exhibit various degrees of skew based on a mode parameter m, 0<=m>=1.

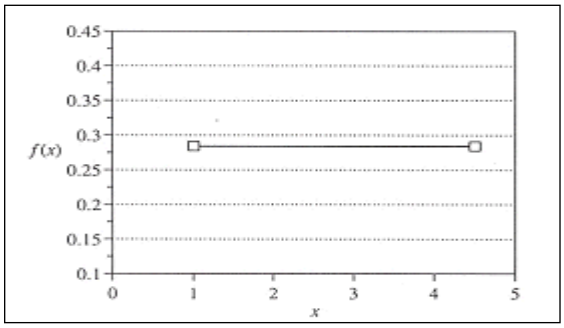




Uniform

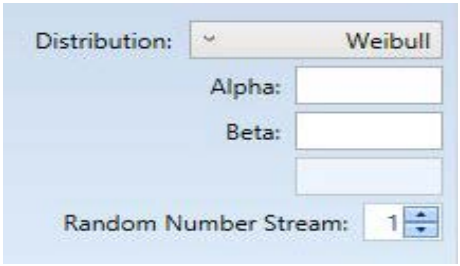
The uniform distribution represents the situation whereby a random number can take values within a finite range with equal probability.

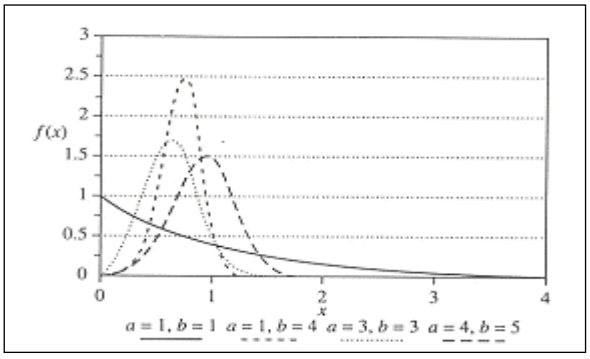




Weibull

The Weibull distribution is used for reliability measures such as the lifetime of components. It is parameterized by two floating values where Alpha > 0 and Beta < 0.





APPENDIX B - TRANSCRIPT WINDOW CONTENT DESCRIPTION

As GENESYS simulates a behavior model, it collects execution data in the transcript window. Each row of data represents a single, discrete event in the simulation run. These events include the start and finish of **Functions** and other constructs within the simulation model. Since a **Function** can appear multiple places within the FFBD, different constructs representing the same **Function** can be executing simultaneously. Moreover, a **Function** construct within a Loop or Iterate construct can execute multiple times during the course of the simulation. Therefore, it is essential to precisely characterize events in the transcript window.

The data is useful for debugging unexpected or undesired simulation results by determining the exact order in which and times at which certain events occurred. The transcript window data can also be used to drive limited analyses of the simulation results by filtering, parsing, counting, adding, and subtracting rows; however, scripts and variables can be used within the model to make this process more efficient (See Section 6).

The contents of each column, from left to right, are described below.

Column 1 – Time

The first column contains a floating-point number showing the simulation time at which the event occurred. Frequently, multiple events occur at the same time, and the list is grouped according to time. The unit of time is not specified but is relative to the Duration attribute of **Functions** and the Delay attribute of **Links** within the simulation model.

Column 2 – Event ID

To understand the contents of this column, a distinction must be drawn between the “scheduled order” of events and the “order scheduled.” The scheduled order is the order in which the events are handled or executed and the order in which they generate their transcript window output. However, before an event can be executed at a scheduled time, it must first be added to the schedule. The order scheduled is the order in which events are added to the schedule, but this may be different from the order in which the events are scheduled to be executed (the scheduled order). For example, when a **Function** *start* event is executed, the corresponding *finish* event is added to the schedule. However, the time at which the *finish* event is executed depends on the Duration of the **Function**. The *finish* of a **Function** with a longer Duration might be scheduled before but executed after the *finish* of a **Function** with a shorter Duration.

While the first column (time) represents the scheduled order, the second column represents the order scheduled. As each primary event is created and added to the schedule, it is assigned a sequential event ID, which is shown in this column. Associated with each primary event may be one or more “auxiliary” events, which are designated by ‘(aux)’ in the second column. An auxiliary event is never scheduled but occurs during the handling of a primary event. An auxiliary event is listed before the primary event that caused it. Primary and Auxiliary events are explained under the description of the contents of Column 4.

Column 3 – Process ID

A Parallel construct of an FBBD in GENESYS represents the simultaneous execution of all its branches. The simulator must be able to simulate multiple, concurrent “threads of execution.” These are called “processes.” A simulator process corresponds to a branch of the simulation model, for example, a branch of a Parallel construct. However, a process is an “execution artifact.” It is not actually part of the model; rather, it represents the execution of a branch in the model. In a simulation, every construct of an FFBD is executed in the context of a process corresponding to its parent branch. Multiple processes may correspond to the same branch in the model. For example, in a Loop or Iterate construct, the same branch may be executed multiple times. A new process is created each time the branch is executed.

Processes are arranged in a parent-child hierarchy corresponding to the arrangement of branches in the simulated FFBD (and the decomposition of its children **Functions**). The root of the process tree corresponds to the main branch of the simulated FFBD. A child process is spawned when the simulation

encounters a Parallel construct, a Selection construct, a Loop construct, an Iterate construct, a multi-exit **Function**, or a **Function** with the Execute Decomposition attribute set to *true*. In the case of a Parallel construct, multiple child processes are generated. While traversing the branch of the nested construct, the main process is suspended and the child process is active. At the end of the traversal of the branch of the nested construct, the child process is terminated and execution of the main process resumes. In the case of a Parallel construct, multiple child processes are active, and constructs on each branch execute in the context of the corresponding process. The parent process resumes when all the child processes terminate or when a child process corresponding to a kill branch terminates.

If the simulated FFBD has multiple levels of nested constructs, then child processes may generate their own children, etc. When each process is generated, it is assigned a sequential hierarchical ID number based on the ID of the parent process in which context it was generated. For example, if the simulation encounters a Parallel construct with three branches in the context of Process 8.9, then the processes generated for the three Parallel branches will receive ID numbers 8.9.1, 8.9.2, and 8.9.3. If the Parallel construct is followed by an Iterate construct, and the Count attribute of the associated **DomainSet** is set to 3, then the process IDs for the three iterations of the branch inside the Iterate construct will be 8.9.4, 8.9.5, and 8.9.6. The second column of the transcript window output reflects the process ID in which context each event is handled.

Column 4 – Event Name

The fourth column of the transcript window output lists the event type. Many events are related to each other and occur in pairs or higher-order groupings. For example, every *finish* event has a corresponding *start* event. However, related events often do not appear in consecutive rows of transcript window output. They are frequently separated in time and by other events. Descriptions of the various event types follow:

Primary Events

start, ***finish***, ***functionTimeout*** – Every time a GENESYS simulation executes an FFBD construct (**Function** construct, Parallel construct, etc.), it logs a *start* event for that construct. If the simulation runs to completion, then it will also finish every construct it starts and log a corresponding *finish* event.

In the case of a **Function** construct, **IF** the following conditions are true:

- It has no decomposition OR
- Its Execute Decomposition attribute is set to *false*
- If a Timeout attribute has been specified for it AND
- If it is triggered by one or more **Items** AND/OR
- It *captures* or *consumes* one or more **Resources**,

THEN a *functionTimeout* event may be logged in lieu of a *finish* event, **IF** the following is true:

After the simulation's flow of control reaches and enables the **Function** construct, the Timeout attribute duration elapses before

1. The triggering Items **AND/OR**
2. The captured or consumed Resources become available.

---

transmittedEvent, ***receivingEvent***, ***receivedEvent*** – Although **Links** are not represented in FFBDs or EFFBDs, a simulation may include **Link** behavior **IF** the following are true:

- An **Item** instance output by a **Function** in the simulation model is carried by a **Link**
- The **Link** is constrained (that is, the Capacity attribute of the **Link** has been assigned a value) **AND**
- The Capacity of the **Link** attribute is non-zero

The relationship between an **Item** instance and a **Link** can be described by two time windows. The first is the time period during which the **Item** instance is being transmitted on one end of the **Link**. The second is the time period during which the **Item** instance is being received on the other end of the **Link**. These time windows may be coincident, overlapping, adjacent, or non-intersecting.

- The *transmittedEvent* occurs at the point in time when the "trailing edge" of the Item instance goes on the **Link.**
- The *receivingEvent* occurs at the point in time when the "leading edge" of the **Item** instance has completely traversed the **Link** and comes off the other end.
- The *receivedEvent* occurs when the trailing edge of the Item instance comes off the **Link**.

Between the time when the leading edge goes on the **Link** and the trailing edge comes off the **Link**, some or the entire **Item** instance is propagating across the **Link** medium.

IF the following are true:

- The Size attribute of the **Item** is relatively large
- The Capacity attribute of the **Link** is relatively large **AND**
- The Delay attribute of the **Link** is relatively short

THEN the *receivingEvent* may occur before the *transmittedEvent*.

Specifically, the leading edge of the Item instance may completely traverse the **Link** (and the Item instance may start coming off the **Link**) before the trailing edge of the **Item** instance has been transmitted by the sending side.

Note that there is no *transmittingEvent* because transmission begins at the same time the **Item** instance is output by a **Function**, i.e. when the *transmittedEvent* and the *receivingEvent* are added to the schedule. However, if either event is scheduled to occur at that same time, then the event is handled immediately and it is never added to the schedule nor logged to the simulator transcript.

The *receivedEvent* is added to the schedule during the handling of the *transmittedEvent*; however, similarly, if the *receivedEvent* occurs at the same time as the *transmittedEvent*, then it is handled immediately and not added to the schedule nor logged to the simulator transcript.

Currently, only one kind of **Link** behavior is supported. The *transmittedEvent* always occurs as soon as the **Item** instance is output and therefore never appears in the simulator transcript. The time of the *receivingEvent* is given by adding to that time the Delay attribute of the **Link** and the Size attribute of the **Item** divided by the Capacity attribute of the **Link**. The *receivedEvent* is always concurrent with the *receivingEvent*. Other types of **Link** behavior will be supported in future releases.

Auxiliary Events

captured, ***released*** – If a **Function** *captures* a **Resource**, then the *captured* event is logged to the transcript window when a GENESYS simulation starts executing a construct representing that **Function**. The **Function** construct cannot start until a sufficient amount of the **Resource** is available. When the simulation finishes executing the **Function** construct, the *released* event is logged. (See also the descriptions of the *produced* and *consumed* events.)

---

enabled – When the flow of control of a GENESYS simulation arrives at a **Function** construct, an *enabled* event is logged to the transcript window. Note that the **Function** construct may not be executed immediately if the **Function** depends upon **Resources** or triggering **Items** that are not available at that time.

---

exitingDecomposition – The decomposition of a **Function** is represented by an FFBD view opened on that **Function**. Function nodes that appear in that FFBD may have their own decompositions. A GENESYS simulation traverses and executes the decomposition of every **Function** construct it encounters where the Execute Decomposition attribute of the Function is set to *true*.

Within the decomposition of a child or descendant **Function** of the main FFBD, if the simulation encounters an Exit construct, then the simulation logs an *exitingDecomposition* event to the transcript window and returns to the parent level, continuing with the construct following the **Function** construct which decomposition it exited.

---

queued, ***dequeued*** – A GENESYS simulation logs a *queued* event to the transcript window when it finishes executing a **Function** construct (with no decomposition or with the Execute Decomposition attribute set to *false*) and the **Function** construct outputs an **Item** instance that triggers another **Function** construct (with no decomposition or with the Execute Decomposition attribute set to *false*) in the simulation model.

If the ***Item*** is carried by a **Link** with a non-zero Capacity attribute, then the *queued* event occurs when the entire Item instance has completely traversed the **Link** from one end to the other.

The *dequeued* event is logged when the simulation starts to execute a Function construct (with no decomposition or with the Execute Decomposition attribute set to *false*) that is triggered by an **Item** of which an instance has been queued.

Even if a **Function** construct has been enabled (see the description of the *enabled* event), execution of the **Function** construct cannot start until triggering Item instances are available.

Once the Item instance has been dequeued, the **Function** construct (or any other **Function** construct triggered by the same Item) cannot start again until another instance of the same **Item** has been queued.

(Note that the *queued* and *dequeued* events apply to triggers. The corresponding events for data stores are *write* and *read*.)

---

produced, ***excessProduced***, ***consumed*** – If a **Function** *consumes* a **Resource,** then the *consumes* event is logged to the transcript window when a GENESYS simulation starts executing a construct representing that **Function**.

The Function construct cannot start until a sufficient amount of the **Resource** is available or has been produced by other Function constructs in the simulation model.

If a **Function** *produces* a **Resource**, then the *produces* event is logged to the transcript window when the simulation finishes executing a construct representing that **Function**.

If a Maximum amount is specified for the **Resource** and a **Function** construct attempts to produce an amount that when added to the amount already available would exceed the maximum allowed, then the *excessProduced* event is also logged.

(See also the descriptions of the *captured* and *released* events.)

---

timeout – A *timeout* event is logged to the transcript window in conjunction with the handling of a *functionTimeout* primary event.

---

transmitting, ***transmitted***, ***backlogged***, ***receiving***, ***received*** – If an **Item** instance is output by a **Function** construct in a simulation model and the Item instance is carried by a **Link** with a non-zero Capacity attribute value, then *transmitting*, *transmitted*, *receiving*, and *received* events will be logged to the transcript window by the interaction of the **Item** instance with the **Link**.

As discussed in the description of the *transmittedEvent*, *receivingEvent*, and *receivedEvent* primary events, the relationship between the **Item** instance and the **Link** can be described by two time windows.

The time period during which the **Item** instance goes on one end of the **Link** is bracketed by the *transmitting* and *transmitted* events.

The time period during which the **Item** instance comes off the other end of the **Link** is bracketed by the *receiving* and *received* events.

The *transmitting* event occurs during the handling of the *finish* primary event for the Function construct that outputs the Item instance.

The *transmitted* auxiliary event occurs during the handling of the *transmittedEvent* primary event. If the *transmittedEvent* is not scheduled or logged because it is concurrent with the *transmitting* event, the *transmitted* auxiliary event is still logged.

The *receiving* auxiliary event occurs during the handling of the *receivingEvent* primary event. If the *receivingEvent* is not scheduled or logged because it is concurrent with the *transmitting* event, the *receiving* auxiliary event is still logged.

The *received* auxiliary event occurs during the handling of the *receivedEvent* primary event. If the *receivedEvent* is not scheduled or logged because it is concurrent with the *transmittedEvent*, the *received* auxiliary event is still logged.

If the **Link** that carries an Item instance does not have sufficient capacity to do so at the time the Item instance is output by a **Function** construct, then the *backlogged* event is logged to the transcript window and the Item instance enters a FIFO (first in, first out) queue.

The sequence of events beginning with *transmitting* then starts the next time sufficient capacity is available on the **Link**.

---

waitingForResources – The *waitingForResources* event is logged to the transcript window between the *enabled* event and the *start* event for a **Function** construct. The *waitingForResources* event occurs after any **Item** instances that trigger the **Function** construct become available.

Note that the Function construct still may not be executed immediately if the **Function** depends upon **Resources** that are not available at that time.

---

write, ***read*** – A GENESYS simulation logs a *write* event to the transcript window when it finishes executing a **Function** construct (with no decomposition or with the Execute Decomposition attribute set to *false*) and the **Function** construct outputs an Item instance that is input to another **Function** construct (with no decomposition or with the Execute Decomposition attribute set to *false*) in the simulation model.

If the Item is carried by a **Link** with a non-zero Capacity attribute value, then the *write* event occurs when the entire Item instance has completely traversed the **Link** from one end to the other.

The *read* event is logged when the simulation starts to execute a **Function** construct (with no decomposition or with the Execute Decomposition set to *false*) that inputs an **Item**. The **Function** construct can start and the *read* event occurs even if it has not been preceded by a corresponding *write* event.

(Note that the *write* and *read* events apply to data stores. The corresponding events for triggers are *queued* and *dequeued*.)

Column 5 – Construct ID

When GENESYS builds a simulation model from a GENESYS behavior model, each construct and branch in the simulation model is assigned a unique ID. The structure of the behavior model can be represented as a hierarchy. The branches of, for example, a Parallel construct can be considered children of the Parallel construct. Other constructs (Function constructs, Parallel constructs, etc.) on that branch can be considered children of that branch. A hierarchical view of the behavior model can be seen in the Execution Hierarchy pane of the timeline elements window. The Execution Hierarchy also shows the unique ID of each construct and branch. The unique IDs are hierarchical numbers corresponding to positions within the hierarchy.

The fifth column of the transcript window output lists the ID of the construct responsible for each event. In some cases (e.g., Parallel constructs, which have no names), it is only possible to determine which construct an event references by correlating the construct ID in this column with the Execution Hierarchy in the timeline elements window. The construct ID, in conjunction with the process ID of an event, can be used to match related events, such as the *start* event associated with a particular *finish* event.

Note that in the simulation model, a multi-exit **Function** is represented by two constructs, a **Function** construct and a “Function Exit Path” construct. The Function Exit Path construct is similar to a Selection construct and can be seen in the Execution Hierarchy immediately after and in the same branch as the corresponding **Function** construct.

Column 6 – Structure

If the event is executing a control construct, then the name of the structure being processed is displayed in this column. If the event is processing an entity, then this column is blank.

Column 7 – Number

If the event is executing an entity, then the number of the entity being processed is displayed in this column. If the event is processing a structure then this column is blank.

Column 8 – Name

If the event is executing an entity, then the name of the entity being processed is displayed in this column. If the event is processing a structure, then this column is blank.

Column 9 – Event Execution Data

The format of the ninth column depends on the type of event listed in the fourth column (event name) and the type of construct that generated the event. The possible execution data formats specific to each kind of event are listed below. Angle brackets are used to designate variable column values, with the text between the brackets describing the information (e.g., <Function number> would indicate the Function number attribute would appear in the column). Single quotation marks delineate literal column values (e.g., ‘to’).

captured, **released**, **produced**, **excessProduced**, **consumed**:
<Function number> <Function name> <amount> ‘to’ <Resource name>
<Function number> <Function name> <amount> ‘from’ <Resource name>

enabled, **waitingForResources**, **functionTimeout**, **timeout**:
<Function number> <Function name>

exitingDecomposition:
'Exit' <Exit name>

queued, **dequeued**:
<Function number> <Function name> <Item name> 'to' <Item name>
<Function number> <Function name> <Item name> 'from' <Item name>

start, **finish**:
'Exit From' <Function number> <Function name>
<Function number> <Function name>
'Iterator'
'Loop'
'Parallel'
'Selection'

transmittedEvent, **receivingEvent**, **receivedEvent**:
<Link number> <Link name>

transmitting, **transmitted**, **backlogged**, **receiving**, **received**:
<Function number> <Function name> <Item name> 'to' <Item name> 'via' <Link number> <Link name>

write, **read**:
<Function number> <Function name> <Item name> 'to' <Item name>
<Function number> <Function name> <Item name> 'from' <Item name>
<Function number> <Function name> 'nil' 'from' <Item name>

Log Messages

One or more lines of the simulation transcript may also be used to record log messages. A log message is drawn from the Log Message attribute of a Function in the simulation model. The value of this attribute, if any, is printed to the Transcript Window when the simulation executes a Function construct representing the Function. The message is logged during the handling of the *start* event. Log messages begin and end with '###' and also include the alias of the Function class and the Function name.

Vitech Corporation
2270 Kraft Drive, Suite 1600
Blacksburg, Virginia 24060
540.951.3322 FAX: 540.951.8222
Customer Support: support@vitechcorp.com
www.vitechcorp.com