IMPRINT

Improved Performance Research Integration Tool

Version 3.0

IMPRINT Pro User Guide

Volume 3
IMPRINT Pro Syntax Reference Manual
March 2009

ALION
SCIENCE AND TECHNOLOGY

COPYRIGHT
© 2003- 2009 Alion Science and Technology. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, photocopying, manual or otherwise, in whole or in part, without the prior written permission of Alion Science and Technology.

TRADEMARKS
Excel, Notepad, Windows, Windows NT, Windows 2000, Windows Me, Windows XP, and Windows Classic are trademarks of Microsoft Corporation.
.NET is a registered trademark of Microsoft Corporation. GoLayout Win is a registered trademark of Northwoods.

All other products and brand names are trademarks or registered trademarks of their respective owners.

Printed in USA

North America:
Alion Science and Technology
4949 Pearl East Circle, Suite 300
Boulder, CO 80301 USA
Phone: (303) 442-6947
Fax: (303) 442-8274
Email: sales@alionscience.com
Website:www.alionscience.com
www.maad.com

Table of Contents

Chapter 1: Introduction 1
Document Conventions 6
Using Online Help 6
Displaying Version Information 6
Technical Support 7

Chapter 2: Expressions 9
Elements in Expression 9
Constants 9
Variables 10
Variable Scoping 11
Macros 11
Comments 12
Commas 12
Return Statements 13
Arithmetic Operators 14
Assignment Operator 15
Adjustment Operators 15
Logical Operators 16
If-else Statements 18
While Statements 19
For Loops 20
Do Statements 21
Switch Statements 21
Foreach Statements 23

Chapter 3: Built-in Macros 25
Macro Syntax and Usage 25
Model Macros 26
Abort 27
Abort(array) 28
Beep 28
CreateEntity 28
Direction 29
Distance 29
Halt 30
Kill 30
Message 30
Pause 32
PrintOutput 32
Random 33
RandomInteger 33
RandomSeed 33

Resume 34
Resume(array) 34
RunNumber 34
SendExternalVariable 35
Start 35
Stop 35
Stop(array) 36
Suspend 36
Suspend(array) 39
ToString 39
TriggerSnapshot 39
WallClock 40
Math Macros 40
Constants 40
Abs 41
Acos 41
Asin 41
Atan 41
Atan2 42
BigMul 42
Ceiling 42
Cos 42
Cosh 43
DivRem 43
Exp 43
Floor 43
IEEERemainder 44
Log 44
Log10 44
Max 44
Min 45
Pow 45
Round 45
Sign 45
Sin 46
Sinh 46
Sqrt 46
Tan 46
Tanh 47
Distribution Macros 47
Bernoulli 48
Beta 48
Binomial 49
DiscreteUniform 49
Exponential 49
ExtremeValueTypeA 50
ExtremeValueTypeB 50

Gamma 50
Geometric 51
InverseGaussian 51
Logistic 51
LogLogistic 51
Lognormal 52
NegativeBinomial. 52
Normal 52
Pareto 52
PearsonType5 53
PearsonType6 53
Poisson 53
Rectangular 53
Triangular 54
Weibull. 54

Chapter 4: Probability Distributions 55
Overview of Probability Distributions 55
References for Probability Distributions 56
Types of Probability Distributions 57
Bernoulli Distribution 57
Beta Distribution 58
Binomial Distribution 58
DiscreteUniform Distribution 59
Exponential Distribution. 59
ExtremeValueTypeA Distribution. 60
ExtremeValueTypeB Distribution. 60
Gamma Distribution. 61
Geometric Distribution 61
InverseGaussian Distribution. 62
Logistic Distribution 62
LogLogistic Distribution 62
Lognormal Distribution. 63
NegativeBinomial Distribution 63
Normal Distribution 64
Pareto Distribution 64
PearsonType5 Distribution 65
Pearson Type 6 Distribution 65
Poisson Distribution. 66
Rectangular (Uniform) Distribution 66
Triangular Distribution 67
Weibull Distribution 67

Glossary 71

Appendix A: Check for Errors Option 87
Using Check for Errors 87

Error Displays 88
Sample Errors and Corrections 89

Index. 95

Volume 3

IMPRINT Pro Syntax Reference Manual

Chapter 1: Introduction

This volume is intended for IMPRINT Pro users that need access to some of the most advanced features of IMPRINT Pro. This volume will help you understand the syntax that IMPRINT Pro provides for you to customize your Mission models in a way that makes them more powerful and flexible.

The syntax in IMPRINT Pro is consistent with the C#, an advanced programming language, If you desire additional information regarding the capabilities of the C# language, you will be able to find more information in a commercial text.

This volume is the third in the IMPRINT Pro User Manual series. The contents of the volumes are outlined below.

Volume 1–IMPRINT Pro Basic Procedures

Chapter 1 Introduction

This chapter contains an overview of IMPRINT Pro, lists the contents of the guide, explains how to use the online help feature, and outlines the conventions used in documentation.

Chapter 2 Installing IMPRINT Pro

This chapter describes system requirements and how to install and start IMPRINT Pro.

Chapter 3 Understanding IMPRINT Pro Analyses

This chapter describes the various elements of a IMPRINT Pro analysis. Sections are arranged around conducting analyses of system missions and equipment. This chapter includes descriptions of the IMPRINT Pro interfaces, data collection, and analysis features. The chapter concludes with a general discussion of the steps involved in creating, debugging, and running IMPRINT Pro analyses.

Chapter 4 Understanding the IMPRINT Pro Window

This chapter presents an overview of the components of the IMPRINT Pro window and how to move, resize, hide, display, close, and dock the window panes.

Chapter 5 IMPRINT Pro Menu Structure

This chapter describes the elements of the IMPRINT Pro menu structure and identifies the capabilities and features available under each element.

Appendix A Technical Description of Stressor Implementation

This appendix contains a discussion of the Evaluation of Human Performance under Diverse Conditions using Modeling Technology

Appendix B Human Performance Micromodels

This appendix contains a list of micromodels available in IMPRINT Pro.

Appendix C

This appendix describes the impact of Training and Personnel Characteristics on taxons.

Appendix D

This appendix outlines the steps required to set up your machine to use Local Server and Remote Server functionality within IMPRINT Pro.

Glossary

The glossary contains a listing of the terms used in IMPRINT Pro.

Volume 2 – Developing Analyses in IMPRINT Pro

Chapter 1 Basic Procedures

This chapter describes how to create, open, and save analyses, export and import analyses and results, and search for text.

Chapter 2 Understanding the IMPRINT Pro Window

This chapter presents an overview of the components of the IMPRINT Pro window and how to move, resize, hide, display, close, and dock the window panes.

Chapter 3 Overview of Analyses

This chapter describes the elements of an IMPRINT Pro Analyses, including Warfighter, Missions, Equipment, and User Stressor.

Chapter 4 Warfighters Data

This chapter describes how to enter the data needed to describe the warfighters that will operate, maintain, supply and support the system.

Chapter 5 Mission Analysis

This chapter describes the IMPRINT Pro Mission module and is divided into several parts, described below.

Subchapter 5.1 Mission Network Diagram

This subchapter describes how to create a mission model network diagram that describes the system mission.

Subchapter 5.2 Working With Missions

This chapter describes the parameters for mission model components: tasks, networks, comments, groups, variables, functions, events, and snapshots. It also describes how to establish performance parameters for the tasks, including time, accuracy, and workload.

Subchapter 5.3 Running the Mission

This chapter describes the settings that control model execution, including speed options. How to check a model for syntax and logic errors is also discussed.

Subchapter 5.4 Mission and Personnel Reports

This chapter describes methods used to analyze the model data and how to create charts displaying the values of variables during execution. How to export and open trace, queue, snapshot and task data files is also discussed.

Chapter 6 Equipment Analysis

This chapter describes the IMPRINT Pro Equipment analysis capability and is divided into several parts, described below.

Subchapter 6.1 Equipment Data

This subchapter describes how to enter the data needed to describe the system equipment, so that IMPRINT Pro can assess maintainability and availability of the system.

Subchapter 6.2 Scenarios

This chapter describes the parameters for operational scenarios: segments, maintainer manning, and sparing.

Subchapter 6.3 Running the Equipment Scenario

This chapter describes the settings that control model execution, including speed options.

Subchapter 6.4 Analyzing Equipment Results

This chapter describes methods used to analyze the results data and how to create charts displaying the values of variables during execution.

Chapter 7 Custom Performance Moderators

This chapter describes how users can create their own performance moderator equations (stressor and training algorithms) that can be used to affect task time and/or accuracy.

Chapter 8 Force Data

This chapter describes how to predict the manpower needed to perform the routine work done by a force unit. Similar to the Define Equipment module in IMPRINT Pro, the Force Data module operates using a stochastic model which relies on various inputs you provide.

Chapter 9 Plugins

This chapter describes how developers can create their own plugins to extend the runtime functionality of the IMPRINT Pro simulator.

Glossary

The glossary contains a listing of the terms used in IMPRINT Pro.

Appendix A Technical Description of Stressor Implementation

This appendix contains a discussion of the evaluation of human performance under diverse conditions using modeling technology.

Appendix B Human Performance Micromodels

This appendix contains a list of micromodels available in IMPRINT Pro.

Volume 3–IMPRINT Pro Syntax Reference Manual

Chapter 1 Introduction

This chapter contains an overview of IMPRINT Pro, lists the contents of the guide, explains how to use the online help feature, and outlines the conventions used in documentation.

Chapter 2 Expressions

This chapter explains the constants, variables, functions, comments, operators, statements, and loops used to create IMPRINT Pro expressions.

Chapter 3 Buit-in Macros

This chapter discusses built-in IMPRINT Pro macros, including Model, Math, and Distribution.

Chapter 4 Probability Distributions

This chapter discusses the twenty probability distributions that are available in IMPRINT Pro.

Appendix

This appendix contains a list of syntax errors and fixes.

Glossary

The glossary contains a listing of the terms used in IMPRINT Pro.

Document Conventions

IMPRINT Pro documentation uses the following terms and typographical conventions:

- File names, variables, and program code are shown in this `typeface`.
- Labeled buttons, menu commands, and menu options are in bold. For example, Click **OK** to close the dialog box.
- Information that the user needs to enter is in Arial Italic. For example, enter *BO90001* in the text box.

Using Online Help

IMPRINT Pro contains extensive online Help to assist you as you work. To display help contents, select **Help** from the **Help** menu.

Displaying Version Information

You can view version information for IMPRINT Pro. You may want to know the version number when you are requesting technical support.

To display version information:

1. From the **Help** menu, select **About**.

   The About IMPRINT Pro dialog box information displays. The version number of the software is indicated.

2. To close the dialog box, click the ☒ in the upper right corner.

Technical Support

You can e-mail any questions you have regarding IMPRINT Pro to the following address:

imprint-info@arl.army.mil

Chapter 2: Expressions

An IMPRINT Pro expression is a calculation, formula, function, or statement that supplies a value or performs an operation. You can use expressions to supply numeric values such as mean times or true/false values such as release conditions. You can also make changes in the state of a model, such as beginning effects.

Elements in Expression

Each expression in IMPRINT Pro must end with a semi-colon. Expressions can include the following elements: constants, variables, functions, comments, commas, return statements, arithmetic operators, assignment operators, adjustment operators, logical operators, and if-else, while, for loops, do, switch, and foreach statements.

Constants

A constant is a quantity with a fixed, unvarying value—usually a number. Often in a model you supply expressions that are constants for numeric task descriptors such as the mean time and standard deviation, or for logical task descriptors that are always true (like the default release condition) or always false.

Constants are also often used in combination with variables and mathematical or logical operators. In the release condition, the value of a variable is compared with a constant. In the mean time and standard deviation, constant values are multiplied by variable values as adjustments.

Expressions that are constants	Expressions that contain constants
`[Release Condition} return 1;`	`{Release Condition} operators > 0;`

Expressions that are constants	Expressions that contain constants
`{Mean Time}` `return 9;`	`{Mean Time}` `15 * (1+FatiqueFactor);`
`{Standard Deviation}` `return 1.5;`	`{Standard Deviation}` `2* (1+Temperature/2);`

Variables

A variable is a string of characters that is assigned a numeric value. Variable names must begin with a letter or underscore and can be followed by zero or more letters, numbers, and underscore characters. Valid examples include abc, _abc, _aA12.

Variable names are case-sensitive, so name and NAME are considered to be different variables.

The Microsoft C# language reserves 76 words for its own use. Do not use any of these as variable names. The keywords include the following:

> abstract, as, base, bool, break, byte, case, catch, char, checked, class, const, continue, decimal, default, delegate, do, double, else, enum, event, explicit, extern, false, finally, fixed, float, for, foreach, goto, if, implicit, in, int, interface, internal, is, lock, long, namespace, new, null, object, operator, out, override, params, private, protected, public, readonly, ref, return, sbyte, sealed, short, sizeof, stackalloc, static, string, struct, switch, this, throw, true, try, typeof, unit, ulong, unchecked, unsafe, ushort, using, virtual, void, and while.

You can change the value assigned to a variable at any time using the assignment operator (=) or one of the adjustment operators (+=, -=, *=, /=). In the following example, the variable Operators (representing the number of operators available to perform the task) decreases by one when the task starts and increases by one when it finishes. You can use a variable in any place that you would use a constant in an expression.

```
//Beginning Effect
Operators -= 1; or Operators = Operators - 1;
//Ending Effect
Operators += 1;  or  Operators = Operators +1;
```

Variable Scoping

Global variables must be defined in the variable list in IMPRINT Pro. Local variables do not need to be defined in the variable list but are defined inside of a code section (for example, Beginning Effect). Local variables can be used to hold temporary values. An example might be when you are using a for loop. Rather than defining a global variable that iterates the loop, you could use a local variable as the loop iterator.

For example,

```
For (int i=0; i<10; i++)
{
     Model.PrintOutput("i: " + i);
}
```

The variable i is defined as an integer at the beginning of the loop and is accessible as long as the loop is running.

Macros

You can call a macro in an expression to perform the procedure associated with the macro or to return a value. All macros must end with a set of parentheses; this includes entity and user-defined macros.

Macro names must begin with a letter or underscore and can be followed by zero or more letters, numbers, and underscore characters. Macro names are case-sensitive so name and NAME are considered to be different macros.

You can use macros that return numeric values in any place that you would use a numeric constant. Macros that perform procedures are generally used as single expressions, as the “then” part of an if-then statement, or as the “do” part of a while loop.

```
//Beginning Effect

if (Clock == 1440)
{
    ModelHalt();
}
```

```
//Mean

Distributions.Poisson(15);
```

Comments

When a model includes long or complex expressions, you may want to add explanatory comments. IMPRINT Pro allows two types of comments:

- Single line comments. In single line comments, the text between // and a carriage return is disregarded.

  `//Macro AVERAGE` calculates the average.

- Multi-line comments. In multi-line comments, the text between /* and */ is disregarded.

  `/*SUMEXPOSE` counts and sums the nonzero elements in the array exposure putting the count in j and the sum in sumexpose*/

Commas

Commas are not allowed in IMPRINT Pro expressions except for in arrays. You can use commas in arrays if you have a two-or-more dimensional array.

For example:

```
int[,] a = new int[12,14];
```

Return Statements

A return statement returns a value so that it can be used in another calculation or evaluation. Return statements must be included in several instances in IMPRINT Pro:

- When an expression in a model contains a variable or value used in other calculations. In this case, the last expression must begin with return.

  For example, the following expression returns the value of b.

  ```
  return b;
  ```

- When logic statements are evaluated in the Release Condition, Decision fields. In this case, a value of true or false must be returned. If a decision is of type probabilistic, a decimal value must be returned.

  For example, the following expression could be used in the Release Condition field.

  ```
  if (x>2)
    {
       return true;
    }
  else
    {
      return false;
    }
  ```

  The task would only execute when `x` was greater than 2.

- When expressions are evaluated in the Mean Time, Standard Deviation, or the Third Data Shaping fields. In this case, a number must be returned.

  For example, the following expression could be used in the Mean Time field.

```
if (Entity.ProductType == 1)
   {
     return 5;
   }
     else
   {
     return 7;
   }
```

When user-defined macros are evaluated (excluding void types) the value of the macro is automatically returned. A return statement is not required.

Arithmetic Operators

You can use the operators in the following table to perform arithmetic operations.

Operator	Purpose	Example
`()`	Grouped operations; Items in parenthesis are executed first.	`(2+3)=7` but `2*(3+1) = 8`
`*`	Multiplication	`2*3 = 6`
`/`	Division	`6/2 = 3`
`%`	Remainder division (also called mod, modular, Modulo, or modulus); only integers can be used with this operation.	`7%2` = remainder of 7/2 = 1 `12%2` = remainder of 12/2 = 0
`-`	Subtraction	`22-11 = 11`
`+`	Addition	`11+22 = 33`
`++`	Increment add 1 to the value of an integer	`x++`
`--`	Decrement subtract 1 from the value of an integer	`x--`

The order of precedence for these operations is from the top to the bottom, that is, exponentiation is performed before multiplication, and so on.

To change the order in which operations are performed, use parentheses. Operations within parentheses are performed first. Inside parentheses, the usual order of operations is maintained.

Assignment Operator

The assignment operator (=) assigns the value on the right of the operator to the variable on the left of the operator. For example, the expression a = 3; assigns the value three to the variable a. It also returns the value three, which is nonzero, so it would be considered a true statement if you used it as a logical condition. However, you would generally not want to use the assignment operator in a logical condition statement.

Be careful not to confuse the assignment operator with the logical equality operator (==). To test for equality, use the logical equality operator (==). This returns a value of one if the value on the right of the operator equals the value on the left, and returns a value of zero if it does not.

`a = 3;` means set a equal to 3

`a == 3` means does the value of a equal 3?

Adjustment Operators

The adjustment operators adjust the value of a variable based on its current value and some other value. These operators provide shorthand for expressions that require repeating the variable name and using the assignment operator, as shown in the following table. You can use either of these forms when you need to adjust the value of a variable.

Long Form	Short Form	Example
variable = variable + number;	variable += number;	If x=2, then `x += 3` would set x to 5
variable = variable – number;	variable -= number;	If x = 2, then `x -= 3` would set x to -1

Long Form	Short Form	Example
variable = variable * number;	variable *= number;	If x = 2, then `x * 2` sets the value x to 4
variable = variable / number;	variable /= number;	If x = 2, then `x /= 2` sets the value of x to 1
variable = variable% number;	variable%= number;	If x = 2, then `x%= 2` sets the value of x to 0

Logical Operators

Use logical operators to compare two numeric values or logical statements. If the comparison is true, the result is equal to one. If the comparison is equal to false, the result is equal to zero. Similarly, any nonzero value is considered true and a zero value is considered false.

Symbol	Comparison	Example
==	Equal to	`a == 3` If the value of a is 3, the expression is true and returns a value of one. If the value of a is anything other than 3, the expression is false and returns a value of zero.
!=	Not equal to	`a != 5` If a is not equal to 5, the expression is true and returns a value of one. If a is equal to 5, the expression is false and returns a value of zero.
<	Less than	`a < 5` If a is less than 5, the expression is true and returns a value of one. If a is greater than 5, the expression is false and returns a value of zero.
<=	Less than or equal to	`a <= 5` If a is less than or equal to 5, the expression is true and returns a value of one. If a is greater than 5, the expression is false and returns a value of zero.

Symbol	Comparison	Example
>	Greater than	`a > 5` If a is greater than 5, the expression is true and returns a value of one. If a is less than 5, the expression is false and returns a value of zero.
>=	Greater than or equal to	`a>=5` If a is greater than or equal to 5, the expression is true and returns a value of one. If a is greater than 5, the expression is false and returns a value of zero.
&&	Logical AND	`a==3 && b==4` If the value of a is 3 and the value of b is 4, the expression is true and returns a value of one. If the value of a is anything other than 3, the expression is false and returns a value of zero.
||	Logical OR	`a==3 || b==4` If the value of a is 3 or the value of b is 4, the expression is true and returns a value of one. If the value of a is anything other than 3, the expression is false and returns a value of zero.
!	Not	`! (a==3)` If a is not equal to 3, the expression is true and returns a value of one. If a is equal to 3, the expression is false and returns a value of zero.

If-else Statements

Sometimes in a model, you may want to perform certain expressions only under specific conditions. IMPRINT Pro uses the if-else statement for in these cases. This statement can be used in several different ways.

- If statements. These take the following format:

  ```
  if (expression) {code to execute};
  ```

  For example,

  ```
  if (x==3)
    {
      y = 5;
    }
  ```

- If-else statements. These take the following format:

  ```
  if (expression to evaluate) {code to execute}
  ```

  ```
  if(x==3)
    {
       y=5;
    }
  else
    {
       z = 3;
    }
  ```

  When you have multiple expressions, you can include the lines in braces, such as in the following:

  ```
  if (x == 5)
    {
       b = 1;
       c = 1;
    }
  if (x == 3)
    {
      y = 5;
    }
  else
    {
      z = 3;
    }
  ```

You can additionally nest if statements, such as in the following example:

```
if(x == 3)
{
   if (y==4)
  {
     z = 3;
  }
}
```

- Cascading If Statements. You can nest if statements inside other if statements. This way you can chain together a sequence of boolean expressions that will be tested one after the other until one evaluates to true.

  For example,

```
if (day==0 | | day==7)

   dayString = "Weekend";

else if (day>1 && day <=6)

   dayString = "Weekday";

else

   dayString ="unknown";
```

While Statements

The while statement executes a block of statements repeatedly until a specified expression evaluates to false. While statements use the following format:

```
while (expression to evaluate) {expressions to
execute}
```

For example, in the following statements, as long as the value of x is less than 10, the value 1 is added to x.

```
while (x<10)
{
x++;
}
```

When a while statement is executed, IMPRINT Pro first checks whether the expression to evaluate is true; if so, it executes the following expression. Then, it checks the initial expression again; if it is still true, IMPRINT Pro executes following expression again. This process continues until the initial expression becomes false. When IMPRINT Pro determines the initial expression is false, it does not execute the following expression again. The initial expression should always be one that eventually becomes false; otherwise you have an infinite loop that does not stop processing.

For Loops

The for loop executes a block of statements repeatedly until a specified expression evaluates to false. A for loop is similar to a while statement, but iterates a set number of times. The for loop is useful when you know how many times a loop needs to be executed. For loops use the following format:

```
for (initializer; condition; iterator value)

{expressions to execute}
```

The initializer sets the initial value of the iterator, the condition controls whether the execution occurs, and the iterator value either increments or decrements the initializer value. The expressions execute as long as condition is true.

For example, in the following statements, as long as the value of j is less than 10, the value 1 is added to j and the number of workers is decreased by one.

```
for (int j=0; j<10; j++)
{
workers --;
}
```

When a for loop is executed, IMPRINT Pro first checks whether the condition is true; if so, it executes the following expression. Then, it checks the condition again; if it is still true, IMPRINT Pro executes following expression again. This process continues until the condition becomes false. When IMPRINT Pro determines the condition is false, it does not execute the following expression again. The condition should always be one that eventually becomes false; otherwise you have an infinite loop that does not stop processing.

Do Statements

The while and for statements both test the boolean expression at the top of the loop. Therefore, if the expression is false on the first test, the body of the loop will not execute. There may be instances where you want the body to run once. For these cases, use the do statement. The syntax of a do statement is the following:

```
do

{expression;}

while (boolean expression);
```

Switch Statements

Switch statements can be used in place of cascading if statements (see "If-else Statements" on page 18). They enable your code to be more efficient and readable. The syntax of a switch statement is the following:

```
switch (control Expression)

{

  case constant Expression:

    statement;
    break;

  case constant Expression2:
```

```
        statement;
        break;

    ...

    default:

        statement;
        break;

    }
```

Switch statements must adhere to the following rules:

- You can only use switch on int and string data types.
- The case labels must be constant expressions such as 3 or “3”.
- The case labels must each have a unique expression; no two can be the same.
- A break; statement is required for each case.
- A default case is allowed.

  For example:

```
string fruit = “apple”;

switch (fruit)
{
    case “banana”:
   {
     Model.PrintOutput (“You ate a banana”);
     break;
   }
    case “apple”:
   {
     Model.PrintOutput (“You ate an apple”);
     break;
   }
    default:
   {
     Model.PrintOutput (“didn’t eat fruit”);
     break;
   }
}
```

Foreach Statements

Sometimes you may want to loop through all the items in an array or some other collection of objects. In this case, you can use a foreach statement. Foreach statements take the following format:

```
foreach (type identifier in expression)
}
   statement;
}
```

For example:

```
foreach (int current in MyArray)
{
   Model.PrintOutput ("current:" + current);
}
```

Chapter 3: Built-in Macros

IMPRINT Pro includes a large number of built-in macros that you can use to calculate values. When IMPRINT Pro encounters the macro, it executes the macro and returns a value. The built-in macros in IMPRINT Pro include the following categories:

- Modeling macros, which perform actions such as starting or stopping tasks, and pausing or stopping model executions.
- Mathematical macros, which perform mathematical operations, such as minimum, maximum, and trigonometric macros.
- Distribution macros, which create probability distributions.

Macro Syntax and Usage

A IMPRINT Pro built-in macro consists of a word identifying the macro type (Model, Math, Distribution, or Animator), followed by a period, and then followed by the macro name.

If the macro requires values supplied to it, include the values in parentheses following the macro name; otherwise, type the parentheses but leave them empty. The values you supply in the parentheses can be numbers, variables, or expressions.

IMPRINT Pro provides assistance when you type a macro name in any expression text box:

- After you type the name of a macro and the left parentheses, IMPRINT Pro automatically displays a tip box with a description of the macro, the variable type for the macro, and the required macro parameters. Click the mouse to close the description.
- After you type the name of the macro type and the following period, IMPRINT Pro automatically displays a list of available macros and properties for your selection. Macros are indicated by a purple, diamond-shape icon and properties are indicated by a grey, rectangular-shape icon. Properties are a type of Model macro that retrieve execution properties such as `RunNumber` and `RandomSeed`. Double-click a macro or property to select it.

You can use a macro anywhere in a model where you need to supply a value or an expression. A few examples are shown in the following list:

```
//Mean Time

Model.Poisson(15);



//Beginning Effect

if (Clock == 1440)
{
  Model.Halt();
{



//Ending Effect

MyMacro( );
```

Model Macros

Model macros control model execution, and can perform actions such as halt, pause, abort, or suspend. All Model macros start with the word Model, followed by a period, and then followed by the macro name, such as `Model.Halt( )`.

For some of the model macros, you supply an attribute for the entity. The attribute can be a default attribute or a user-defined attribute. Default attributes consist of the following:

- `Entity.Duration` records the task execution time for the entity.
- `Entity.Tag` records the tag ID number of the entity. The `Entity.Tag` value is an integer that records the identity of each entity when multiple entities travel through a network.
- `Entity.Group` is similar to the `Entity.Tag` attribute, but is used to control groups of tasks.

For example, the aircraft in a simulation are given assigned tag values, but have scanning tasks that are assigned to a tank commander, gunner, and the loader. When an aircraft is detected, you want to stop the scanning tasks and react to the hostile aircraft. Using the attribute `Entity.Group`, you can label the tank commander's scanning task as `Entity.Group = 1`, the gunner as `Entity.Group = 2`, and the loader as `Entity.Group = 3`. Then you can control whether the group of tasks are active through group attributes. As with the `Entity.Tag` attribute, the `Entity.Group` value is carried through the simulation.

- `Entity.ID` records the current task number the entity is in. The task number is assigned to the task by IMPRINT Pro, but you can change the number in the task ItemID box.
- `Entity.Time` contains the closest scheduled event time for the entity.
- `Entity.Type` contains the event type for which the entity is scheduled. These can be one-time events or they can repeat at regular intervals.

Abort

Syntax: Model.Abort*("key", value*)

Description: Aborts all entities matched by the key-value pair. The key parameter can be a default entity attribute or a user-defined attribute.

Example: `Model.Abort("Tag",3)` aborts any entities with a `tag` value of 3.

`Model.Abort("ID", 2)` aborts any entities in task ID 2.

Abort(array)

Syntax: Model.Abort("*key array", value array*)

Description: Aborts all entities matched by the key-value pair. The key parameter can be a default entity attribute or a user- defined attribute.

Example: If key[0] = "Group," key[1] = "Tag," val[0] = 3, and val[1] =12, then `Model.Abort(key, val)` aborts any entities with a group value of 3 and a tag value of 12.

If key[0] = "Group," key[1] = "ID," val[0] = 3, and val[1] =12 then `Model.Abort("Tag",3)` aborts any entities with a tag value of 3. `Model.Abort("ID", 2)` aborts any entities in task ID 2.

Beep

Syntax: Model.Beep(*number*)

Description: Causes a beeping noise during model execution. You indicate the number of beeps.

Example: `Model.Beep(3)` causes three beeps to sound.

CreateEntity

Syntax: Model.CreateEntity( )

Description: Creates an entity with attributes that are all set to zero.

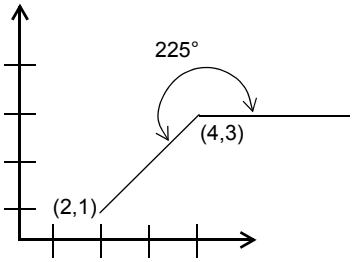
Example: `Entity TemporaryEntity=CreateEntity( )` points the entity variable `TemporaryEntity` to a new entity with attributes that are all equal to zero.

Direction

Syntax: Model.Direction(*x1*, *y1*, *x2*, *y2*)

Description: Returns the direction (in degrees) between two points (x1,y1) and (x2,y2).

Example: `Model.Direction (4,3,2,1)` returns 225.

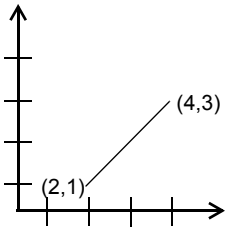


Distance

Syntax: Model.Distance (*x1*, *y1*, *x2*, *y2*)

Description: Returns the distance between two points (x1,y1) and (x2,y2)

Example: `Model.Distance(4,3,2,1)` returns 2.82 (approximately)



Halt

Syntax: Model.Halt( )

Description: Terminates the simulation at the next safe stopping place and sends simulation completion information to the Output window. Also see "Kill" on page 30.

Example: `Model.Halt( )`

Kill

Syntax: Model.Kill( )

Description: Terminates the model execution immediately. Also see "Halt" on page 30.

Example: `Model.Kill( )`

Message

Syntax: Model.Message(*expression to evaluate*, *value 1*, *value 2*, *value 3*)

Message(*value*)

Description: Displays a dialog box with the specified information.

Example: `Model.Message(i==3,x,y,z)` displays a dialog box with the values for x, y, and z if i is equal to 3.

`Model.Message ("I'm Here")` displays a dialog box with the phrase "I'm Here".

Any message you add to your model using the Model.Message macro displays in a dialog box which pops up at the time the macro is called in your model run. The message will temporarily suspend the model run until it is acknowledged by clicking the **OK** button on the dialog. If your model contains numerous model.message calls and you wish to suspend all messages globally rather than clicking OK to each one, you may use one of the following two methods:

To globally suspend all model.message calls in your model - method

1:

1. Define a new variable called “messagesOn” or a similar variable.

2. Somewhere in your model, set this variable to a nonzero value (int or real) or else to “true” (if bool) if you want your messages to display during the model run.

3. In every Model.Message expression you have in your Pro model, four comma-separated items must follow in parentheses:

   - item 1. **messagesOn** (or your corresponding variable you create in step 1 above). If this variable evaluates to true, then your message reporting items 2,3, and 4 will display. If it evaluates to false, your message will not display.)

   - item 2. Can be anything you want to print in the message window.

   - item 3. Can be anything you want to print in the message window.

   - item 4. Can be anything you want to print in the message window.

   For this method to work, you must have 4 comma-separated items in the Model.message line. Here is an example:

   messageOn = 1.

   Model.message(messageOn, “This is the first line of text”, “This is the second line of text”, “This is the third line of text”);

To globally suspend all model.message calls in your model - method 2:

As an alternative, you can write a simple macro which would enable Model.Message only if a global variable were “true”, and then call the Macro in place of Model.Message.

1. Add a new variable, for example “`MessageEnabled`”. Make the type of this variable boolean. Set the value of this variable to true or false as required.

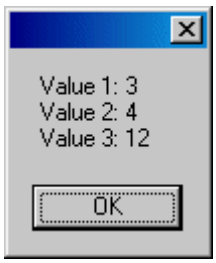
2. Create a new macro called `MyModelMessageMacro`. Double-click the macro in the tree to open it up. In the **Macro Code** window, enter the following code:

   if (MessagesEnabled==true)

```
{
    Model.Message(message);
}
```

3. Click the Parameters tab of the Macro window and add a parameter called “message”. Make the type of this parameter “string”.

   For any instances in your model where messages would have normally been called by `Model.Message(<message>)`, substitute this code with a call for the macro as follows: `MyModelMessageMacro(<message>)` where “message” is the message to display.



Example of message dialog box showing value 1, value 2, and

Pause

Syntax: Model.Pause( )

Description: Pauses the model execution. This macro is useful for debugging models. Place the Pause macro in a task before a known error and then run the model. The model executes in a single-step fashion immediately before the error occurs.

Example: `Model.Pause( )`

PrintOutput

Syntax: Model.PrintOutput(*value*)

Description: Writes a message to the Output window.

Example: `Model.PrintOutput (“In task 2 Beginning effect”)`

Random

Syntax: Model.Random(*minimum*, *maximum*)

Random( )

Description: Returns a random number between the two numbers you supply or returns a random number between 0 and 1, including 0 and excluding 1.

Example: `Model.Random(2,4)` returns a random number between the numbers 2 and 4.

`Model.Random( )` returns a random number between 0 and 1, including 0 and excluding 1.

RandomInteger

Syntax: Model.RandomInteger(minimum, maximum)

Description: Returns a random integer between the minimum and maximum numbers that you supply.

Example: `Model.RandomInteger(1,4)` returns a random integer between the numbers 1 and 4.

RandomSeed

Syntax: Model.RandomSeed

Description: Returns the value of the current random number seed. Do not use parentheses after `RandomSeed`.

Example: In the example `MySeed = Model.RandomSeed` the variable `MySeed` would contain the current seed. The same model run with this seed will produce the same results.

Resume

Syntax: Model.Resume(“*key”, value*)

Description: Resumes all entities matched by the key-value pair. The key parameter can be a default entity attribute or a user-defined attribute.

Example: `Model.Resume (“Tag”, 3)` resumes any entities with a tag value of 3.

`Model.Resume(“ID”, 2)` resumes any entities in task ID 2.

Resume(array)

Syntax: Model.Resume(“*key array”, value array*)

Description: Resumes all entities matched by the key-value pair. The key parameter can be a default entity attribute or a user-defined attribute.

Example: If key[0] = “Group,” key[1] = “Tag,” val[0] = 3, and val[1] = 12, `Model.Resume(key, val)` resumes any entities with a group value of 3 and a tag value of 12.

If key[0] = “Group,” key[1] = “ID,” val[0] = 3, and val[1] = 12, then `Model.Resume (key,val)` resumes any entities with a group value of 3 in task ID 12.

RunNumber

Syntax: Model.RunNumber

Description: Returns the current run number. Do not use parentheses after `RunNumber`.

Example: In the example `MyRun = Model.RunNumber` the variable `MyRun` will store the current run number. This is useful for snapshots of multiple runs.

SendExternalVariable

Syntax: Model.SendExternalVariable (*variable name*)

Description: Sends a variable to an external connected program. This only occurs if IMPRINT Pro is started or "owned" by another application.

Example: Model.SendExternalVariable(MyRun)

Start

Syntax: Model.Start(*ID, tag number*)

Start(*Entity Variable*)

Description: Starts the entity specified by the tag number in the specified task.

Example: `Model.Start (2,2)` starts the entity with the tag value 2 in task ID 2.

TempEnt=Model.CreateEntity( ); TempEnt.Tag = 3; TempEnt.ID = 2; Start(TempEnt);

starts the entity with the tag value 3 in task ID 2

Stop

Syntax: Model.Stop("*key", value*)

Description: Stops all entities matched by the key-value pair. The key parameter can be a default entity attribute or a user-defined attribute.

Example: `Model.Stop ("Tag", 3)` stops any entities with a tag value of 3.

`Model.Stop("ID", 2)` stops any entities in taskID 2.

Stop(array)

Syntax: Model.Stop("*key array", "value array*")

Description: Stops all entities matched by the key-value pair. The key parameter can be a default entity attribute or a user- defined attribute.

Example: If key[0] = "Group," key[1] = "Tag," val[0] = 3, and val[1] = 12, then `Model.Stop(key, val)` suspends any entities with a group value of 3 and a tag value of 12.

If key[0] = "Group," key[1] = "ID," val[0] = 3, and val[1] = 12, then `Model.Stop (key,val)` stops any entities with a group value of 3 in task ID 12.

Suspend

Syntax: Model.Suspend("*key", value*)

Description: Suspends all entities matched by the key-value pair. The key parameter can be a default entity attribute or a user- defined attribute.

Example: `Model.Suspend ("Tag", 3)` suspends any entities with a tag value of 3.

`Model.Suspend("ID", 2)` suspends any entities in task ID 2.

If you wish to suspend one or more entities in a task contained by a function, and you wish to use "ID" as your determining attribute, keep in mind that the ID of any task existing at the function level takes the form of a string. In other words whereas two tasks which are added to the top level of your network might have task IDs of "1" and "2", tasks added within a function located at the top level of could have task IDs of "3_1" and "3_2". These IDs imply the tasks are the first and second nodes added to a function which is third element in the level above. In this case the Model.Suspend command syntax changes as follows:

`Model.Suspend("ID", "3_2")` suspends any entities in task ID 2.

If you encounter any issues with the suspension of entities in your model, you may use a PrintOutput statement to help determine which entities are currently suspended. The following Model.Suspend statement may be used:

```
int x = Model.Suspend("ID", "1");
Model.PrintOutput(x);
```

The first line suspends the entities belonging to the task having the attributes described in the parentheses. By adding the "int x = "to the beginning of the statement, it also assigns the integer from the Model.suspend statement to the local variable "x" which is essentially the tag value of the entity being suspended. The second line of code added is a printout of the suspended entity. If the print statement in the output window reports "1", then any entity in the task having an ID of 1 is suspended. If the print statement reports "0", then no tasks are being suspended.

If you wish to suspend a particular entity within a specific task, the following suspend commands may be used:

Beginning Effect

```
1  string[] key = new string[2];
2  object[] value = new object[2];
3  key[0]="ID";
4  key[1]="Tag";
5  value[0]="1";
6  value[1]=0;
7  int x = Model.Suspend(key,value);
8  Model.PrintOutput(x);
```

where lines 1 and 2 declare two new two-dimensional arrays (“key” and “value”) whose values will be passed through the Model.Suspend statement. The two array values of “key “ are assigned “ID” and “Tag”, respectively, to allow you to suspend an entity based on either its ID value or its Tag value depending on the first value passed in the statement. The two array values of “value” are set to “0” and “1” as an example. These values will determine which task ID or entity will be suspended. In the case of key=1 and value=1 all entities within Task ID 1 are suspended In the case of key=2 and value=1 only the entity which has a Tag value of one will be suspended.

Suspend(array)

Syntax: Model.Suspend("*key array", value array*)

Description: Suspends all entities matched by the key-value pair. The key parameter can be a default entity attribute or a user-defined attribute.

Example: If key[0] = "Group," key[1] = "Tag," val[0] = 3, and val[1] = 12, then `Model.Suspend(key, val)` suspends any entities with a group value of 3 and a tag value of 12.

If key[0] = "Group," key[1] = "ID," val[0] = 3, and val[1] = 12 then `Model.Suspend (key,val)` suspends any entities with a group value of 3 in task ID 12.

ToString

Syntax: Model.ToString( )

Description: Converts a value to a string.

Example: `Model.ToString( )`

TriggerSnapshot

Syntax: Model.TriggerSnapshot(*snapshot ID*)

Description: This macro executes the snapshot with the indicated ID (string value). The macro returns true if the snapshot was executed and returns false if the snapshot was not executed. This macro is an alternate way to make snapshots occur if the default triggers do not fulfill your needs.

Example: Model.TriggerSnapshot(Snap1)

WallClock

Syntax: Model.WallClock(*type*)

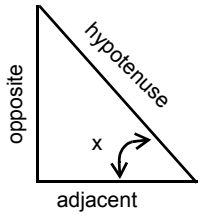
Description: Returns the number of seconds or minutes since the program started running. Use the value 0 for seconds and use the value 1 for minutes.

Example: `Model.WallClock(1)` returns the number of minutes since the model started running.

Math Macros

Mathematical macros are specific to mathematical operations, such as minimum, maximum, and trigonometric functions. All Math functions start with the word Math, followed by a period, and then followed by the function name, such as `Math.Cos( )`.

The following diagram shows the parameters used in trigonometric functions.



Constants

The Mathematical class of macros contains two constants:

- E represents the natural logarithmic base specified by the constant e. You call this constant with the phrase `Math.E`.
- Pi represents the ratio of the circumference of a circle to its diameter and is specified by the constant Π. You call this constant with the phrase `Math.Pi`.

Abs

Syntax: Math.Abs(*value*)

Description: Returns the absolute value of (value).

Example: `Math.Abs(3)` returns the value 3

Acos

Syntax: Math.Acos(*value*)

Description: Returns the angle, in degrees (0-180), whose cosine is (value).

Example: `Math.Acos(adj/hyp)` returns x

Asin

Syntax: Math.Asin(*value*)

Description: Returns the angle, in degrees (90-270), whose sine is (*value*).

Example: `Math.Asin(opp/hyp)` returns x

Atan

Syntax: Math.Atan(*value*)

Description: Returns the angle, in degrees (90-270), whose tangent is (*value*).

Example: `Math.Atan(opp/adj)` returns x

Atan2

Syntax: Math.Atan2(*value 1, value 2*)

Description: Returns the angle whose tangent is the quotient of two specified numbers.

Example: `Math.Atan2(.5,1)` returns the value 2.21429744

BigMul

Syntax: Math.BigMul(*int a, int b*)

where a and b are 32-bit integers

Description: Returns the multiplication of a and b as a long integer(64-bit).

Example: `BigMul(2, 3)` returns 6

Ceiling

Syntax: Math.Ceiling(*value*)

Description: Returns the largest whole number greater than or equal to the specified number.

Example: `Math.Ceiling(5.2)` returns 5

Cos

Syntax: Math.Cos(*value*)

Description: Returns the cosine of the angle value you supply. Enter the angle in radians. To convert degrees to radians, multiple by $\Pi/180$.

Example: `Math.cos(x)` returns the quotient adj/hyp

Cosh

Syntax: Math.Cosh(*value)*

Description: Returns the hyperbolic cosine of the specified angle. Enter the angle in radians. To convert degrees to radians, multiple by Π/180.

Example: `Math.Cosh(x)` returns the quotient adj/hyp.

DivRem

Syntax: Math.DivRem(*value1*, *value 2*)

Description: Division remainder. Returns the remainder from the division operation. Only integers can be used. For floating point division, "IEEERemainder" on page 44.

Example: `Math.DivRem(3,2)` returns the value 1

Exp

Syntax: Math.Exp(*value*)

Description: Returns e raised to the specified power.

Example: `Math.Exp(2)` returns the value 7.3890561

Floor

Syntax: Math.Floor(*value*)

Description: Returns the smallest whole number less than or equal to the specified number.

Example: `Math.Floor(3.8)` returns the value 3

IEEERemainder

Syntax: Math.IEEERemainder(dividend, divisor)

Description: Floating point division remainder (can be used to calculate division error).

Example: `Math.IEEERemainder(1.797693e+308,3.00)` returns -1.995840e+292.

Log

Syntax: Math.Log(*value*)

Description: Returns the natural logarithm of a specified number. *Value* must be greater than zero.

Example: `Math.Log(10)` returns the value 1

Log10

Syntax: Math.Log10(*value*)

Description: Returns the base 10 logarithm of a specified number. *Value* must be greater than zero.

Example: `Math.Log10(107)` returns the value 7

Max

Syntax: Math.Max(*value1*, *value2*)

Description: Returns the maximum of the two values you supply.

Example: `Math.Max(1,2)` returns the value 2

Min

Syntax: Math.Min (*value1*, *value2*)

Description: Returns the minimum of the two values you supply.

Example: `Math.Min(1,2)` returns the value 1

Pow

Syntax: Math.Pow (*value*, *power*)

Description: Returns the value of a specified number raised to the specified power.

Example: `Math.Pow(2,3)` returns the value 8

Round

Syntax: Math.Round (*value1*)

Description: Returns the number nearest to the specified value. Rounds value up or down to the nearest whole number. If fractional part is >= 0.5, value is rounded up (away from 0). Otherwise, it is rounded down (toward 0).

Example: `Math.Round(2.5)` returns the value 2

Sign

Syntax: Math.Sign(*value*)

Description: Returns a value indicating the sign of the number. If the value is less than zero, -1 is returned. If the value is equal to zero, 0 is returned. If the value is greater than zero, a 1 is returned.

Example: `Math.Sign(943839)` returns the value 1

Sin

Syntax: Math.Sin(*value*)

Description: Returns the sine of the angle you supply. Enter the angle in radians. To convert degrees to radians, multiply by Π/180.

Example: `Math.Sin(1.2)` returns the value 0.932039086

Sinh

Syntax: Math.Sinh(*value*)

Description: Returns the hyperbolic sine of specified angle. Enter the angle in radians. To convert degrees to radians, multiply by Π/180.

Example: `Math.Sinh(1.2)` returns the value 1.50946136

Sqrt

Syntax: Math.Sqrt(*value*)

Description: Returns the square root of the specified number.

Example: `Math.Sqrt(4)` returns the value 2

Tan

Syntax: Math.Tan(*value*)

Description: Returns the tangent of the angle you supply. Enter the angle in radians. To convert degrees to radians, multiply by Π/180.

Example: `Math.Tan(1.2)` returns the value 2.57215162

Tanh

Syntax: Math.Tanh(*value*)

Description: Returns the hyperbolic tangent of the specified angle. Enter the angle in radians. To convert degrees to radians, multiply by Π/180.

Example: `Math.Tanh(1.2)` returns the value 0.833654607

Distribution Macros

Distribution macros are specific to probability macros. IMPRINT Pro contains 20 distribution macros that return a random number from a probability distribution. All Distribution macros start with the word Distributions, followed by a period, and then followed by the macro name, such as `Distributions.Beta`.

Additional details on distribution macros are provided in Chapter 4: "Probability Distributions." For specific details regarding the probability distributions, consult a statistics text.

Distribution macro parameters include the following:

- **Psuccess**. The probability of success.
- **Successes**. The number of successes.
- **Mean**. The mean value.
- **Mode**. The most common or frequent value.
- **Standard deviation** (square root of the arithmetic average of the squares of the deviations from the mean).
- **Ntrials**. The number of trials.
- **Minimum**. The minimum value to be used in the distribution.
- **Maximum**. The maximum value to be used in the distribution.

- **Scale**. The scale of measurement of the values in the range of distribution (for more details, see "Overview of Probability Distributions" on page 55).
- **Shape, Shape 1, Shape 2**. The basic form or shape of a distribution within the general family of distributions of interest (for more details, see "Overview of Probability Distributions" on page 55).

Bernoulli

Syntax: Distributions.Bernoulli(*psuccess*)

Description: Returns a random integer from a Bernoulli distribution. You supply the probability of success (*psuccess*) to define the distribution. The value you supply for *psuccess* should be greater than or equal to zero and less than 1.

Example: `Distributions.Bernoulli(.1)`

Beta

Syntax: Distributions.Beta(*mean*, *standard deviation*)

Description Returns a random number from a Beta distribution. You supply the mean and standard deviation to define the distribution.

Example: `Distributions.Beta(4,1)`

Binomial

Syntax: Distributions.Binomial(*ntrials*, *psuccess*)

Description: Returns a random integer from a binomial distribution. You supply the number of trials (*ntrials*) and the probability of success (*psuccess*) values to define the distribution. The number returned represents the number of successes that occur during the trials; this number can be any integer between and including zero and *ntrials*. The value you supply for *psuccess* should be greater than or equal to zero and less than 1.

Example: `Distributions.Binomial(5,.1)`

DiscreteUniform

Syntax: Distributions.DiscreteUniform(*minimum*, *maximum*)

Description: Returns a random integer from a Discrete Uniform distribution. You supply the minimum and maximum values to define the distribution.

Example: `Distributions.DiscreteUniform(4,1)`

Exponential

Syntax: Distributions.Exponential(*mean*)

Description: Returns a random number from an exponential distribution. You supply the mean value to define the distribution. A standard deviation is not used for this macro.

Example: `Distributions.Exponential(4)`

ExtremeValueTypeA

Syntax: Distributions.ExtremeValueTypeA(*mean*, *standard deviation*)

Description: Returns a random number from an Extreme Value Type A distribution. You supply the mean and standard deviation to define the distribution.

Example: `Distributions.ExtremeValueTypeA(4,1)`

ExtremeValueTypeB

Syntax: Distributions.ExtremeValueTypeB(*mean*, *standard deviation*)

Description: Returns a random number from an Extreme Value Type B distribution. You supply the mean and standard deviation to define the distribution.

Example: `Distributions.ExtremeValueTypeB(4,1)`

Gamma

Syntax: Distributions.Gamma(*mean*, *standard deviation*)

Gamma(*shape*)

Description: Returns a random number from the Gamma distribution, using either the mean and standard deviation or the shape factor.

Example: `Distributions.Gamma(4,1)`

`Distrubution.Gamma(4)`

Geometric

Syntax: Distributions.Geometric(*mean*)

Description: Returns a random number from a Geometric distribution. You supply the mean to define the distribution.

Example: `Distributions.Geometic(4)`

InverseGaussian

Syntax: Distributions.InverseGaussian(*mean*, *standard deviation*)

Description: Returns a random number from an Inverse Gaussian distribution. You supply the mean and standard deviation to define the distribution.

Example: `Distributions.InverseGaussian(4,1)`

Logistic

Syntax: Distributions.Logistic(*mean, standard deviation*)

Description: Returns a random number from a Logistic distribution. You supply the mean and standard deviation to define the distribution.

Example: `Distributions.Logistic(4,1)`

LogLogistic

Syntax: Distributions.LogLogistic(*scale*, *shape*)

Description: Returns a random number from a LogLogistic distribution. You supply the scale and shape factors to define the distribution.

Example: `Distributions.LogLogistic(4,1)`

Lognormal

Syntax: Distributions.Lognormal(*mean*, *standard deviation*)

Description: Returns a random number from a lognormal distribution. You supply the mean and standard deviation values to define the distribution.

Example: `Distributions.Lognormal(4,1)`

NegativeBinomial

Syntax: Distributions.NegativeBinomial(*successes*, *psuccess*)

Description: Returns a random number (integer) from a Negative Binomial distribution. You supply the number of successes value and the probability of success to define the distribution.

Example: `Distributions.NegativeBinomial(4,1)`

Normal

Syntax: Distributions.Normal(*mean*, *standard deviation*)

Description: Returns a random number from a normal distribution. You supply the mean and standard deviation values to define the distribution.

Example: `Distributions.Normal(4,1)`

Pareto

Syntax: Distributions.Pareto(*minimum*, *scale*)

Description Returns a random number from a Pareto distribution. You supply the minimum value scale to define the distribution.

Example: `Distributions.Pareto(4,1)`

PearsonType5

Syntax: Distributions.PearsonType5(*scale*, *shape*)

Description: Returns a random number from a Pearson Type 5 distribution. You supply the scale and shape factors to define the distribution.

Example: `Distributions.PearsonType5(4,1)`

PearsonType6

Syntax: Distributions.PearsonType6(*scale*, *shape1*, *shape2*)

Description: Returns a random number from a Pearson Type 6 distribution. You supply the scale and shape factors to define the distribution.

Example: `Distributions.PearsonType6(3, 4,1)`

Poisson

Syntax: Distributions.Poisson(*mean*)

Description Returns a random number from a Poisson distribution. The number returned represents the number of independent events that occur in a particular time interval. The mean value you supply defines the average number of events expected in such time intervals.

Example: `Distributions.Poisson(4)`

Rectangular

Syntax: Distributions.Rectangular*(mean, minimum*)

Description: Returns a random number from the Rectangular distribution. You supply the mean and minimum values to define the distribution.

Example: `Distributions.Rectangular(4,1)`

Triangular

Syntax: Distributions.Triangular(*mode*, *minimum*, *maximum*)

Description: Returns a random number from a Triangular distribution. You supply the mode, minimum, and maximum values to define the distribution.

Example: `Distributions.Triangular(4,1, 9)`

Weibull

Syntax Distributions.Weibull(*scale*, *shape*)

Description: Returns a random number from a Weibull distribution. You supply the scale and shape factors to define the distribution.

Example: `Distributions.Weibull(4,1)`

Chapter 4: Probability Distributions

To help simulate task execution times as realistically as possible, IMPRINT Pro randomly generates the execution times for each task using a probability distribution. When you select a time distribution for a task, IMPRINT Pro uses the distribution to generate random execution times that occur in the pattern predicted by the distribution.

A probability distribution defines how frequently a particular value is likely to occur in a set of observations. For example, in a Normal distribution, values close to the mean are likely to occur, while values far from the mean are not. If you define task times to be distributed normally with a mean of 40 minutes and a standard deviation of 5, the task would usually execute with a time between 30 and 50 minutes.

IMPRINT Pro provides 20 basic and advanced probability distributions for your use.

Overview of Probability Distributions

Probability distributions can be discrete or continuous.

- **Discrete** distributions return integer values and are primarily used in simulating numbers of random events and not in simulating times. Discrete probability distributions provided by IMPRINT Pro include Bernoulli, Binomial, Discrete Uniform, Geometric, Negative Binomial, and Poisson.

- **Continuous** distributions include mean-standard deviation distributions, scale-shape distributions, and minimum value distributions.

  Mean-standard deviation distributions use mean and/or standard deviation as inputs for the distribution. Mean-standard continuous probability distributions provided by IMPRINT Pro include Beta, Exponential, Extreme Value Type A, Extreme Value Type B, Gamma, Inverse Gaussian, Logistic, Lognormal, and Normal distributions.

Scale-shape distributions use standard scale and shape parameters as inputs for the distribution (see the following page for a description of these parameters). These distributions can have undefined means and standard deviations or are not trivial to convert to a mean and standard deviation-dependent function. Scale-shape continuous probability distributions provided by IMPRINT Pro include LogLogistic, Pearson Type 5, Pearson Type 6, and Weibull.

Minimum value distributions use a minimum and/or maximum in addition to mean and/or scale as inputs for the distribution. Minimum value continuous probability distributions provided by IMPRINT Pro include the Pareto, Rectangular, and Triangular.

The equations defining the probability distributions use several parameters. In addition to the more common statistical variables such as mean and standard deviation, the following parameters are used:

- A **scale** parameter, determines the scale of measurement of the values in the range of distribution. A change in the scale parameter compresses or expands the distribution without altering its basic form.
- A **shape** parameter, determines the basic form or shape of a distribution within the general family of distributions of interest. A change in the shape parameter generally alters the properties of a distribution, such as skewness, more than a change in location or scale. Some distributions do not have a shape parameter, while others may have several.

References for Probability Distributions

The following section describes the probability distributions provided by IMPRINT Pro. The descriptions are used with the permission of Stat::Fit and are modified from the following sources:

The descriptions of probability distributions in this help system are used with the permission of Stat::Fit and are modified from the following sources:

- Johnson, N.L., Kotz, Samuel, Balakrishnan, 1994, “Continuous Univariate Distributions, Volume 1”: John Wiley & Sons.
- Johnson, N.L., Kotz, Samuel, Kemp, A.W., 1992, Univariate Discrete Distributions: John Wiley & Sons.

- Law, A.M., and Kelton, W.D., 1991, Simulation Modeling & Analysis: McGraw-Hill.
- Shooman, M.L., 1990, Probabilistic Reliability: An Engineering Approach.

Types of Probability Distributions

IMPRINT Pro provides 21 basic and advanced probability distributions for your use.

Bernoulli Distribution

Syntax: Distributions.Bernoulli(*psuccess*)

Applications: The Bernoulli distribution models the number of successes that can occur in a single trial with probability of success *psuccess*. This distribution best describes situations where a “trial” is made, resulting in either success or failure. Examples include tossing a coin or modeling the success or failure of a procedure. A Bernoulli trial is an experiment in which only two outcomes are possible: success, with probability p, and failure, with probability 1-p.

Description: The Bernoulli distribution is a discrete distribution that is bounded by [0, n]. The Bernoulli distribution is a Binomial distribution with the number of trials = 1.

Beta Distribution

Syntax: Distributions.Beta (*mean, standard deviation*)

Applications: The Beta distribution provides distributions of random proportion and is used for models in the absence of data. Examples include the proportion of defective items in a shipment, the time to complete a task in a network, dissipation rate in breakage models, construction duration, and tool wear.

Description: The Beta distribution is a continuous distribution that has both upper and lower finite bounds. Because many real situations can be bounded in this way, the Beta distribution can be used empirically to estimate the actual distribution before much data are available.

Binomial Distribution

Syntax: Distributions.Binomial(ntrials, psuccess)

Applications: The Binomial distribution models the number of successes that can occur in a sequence of ntrials with probability of psuccess in each trial. For example, if you test a product part five times and each test has a ten percent probability of success, the number of successes is usually zero or one. If each test has a fifty percent probability of success, the number of successes would usually be two or three.

The Binomial distribution is used extensively in games, but is also useful in genetics, sampling of defective parts in a stable process, and other event sampling tests where the probability of the event is known to be constant or nearly constant.

Description: The Binomial distribution is a discrete distribution that is bounded by [0, n]. Typically, it is used where a single trial is repeated over and over.

DiscreteUniform Distribution

Syntax: Distributions.DiscreteUniform(*minimum, maximum*)

Applications: The Discrete Uniform distribution is used where there is random occurrence with several possible outcomes, each of which is equally likely.

Description: The Discrete Uniform distribution is a discrete distribution that is bounded on [*minimum*, *maximum*] with constant probability at every value on or between the bounds. It arises when an event can have a finite and equally probable number of outcomes.

Exponential Distribution

Syntax: Distributions.Exponential(*mean*)

Applications: The Exponential distribution is frequently used to represent the time between random occurrences. Examples include the time between arrivals at a specific location in a queuing model or the time between failures in reliability models. The Exponential distribution is also used to represent times between independent events that occur at a constant rate, such as the time between arrival of aircraft at airports, between arrival of targets in a field of view, or between accidents at a maintenance depot. Typically, Exponential distributions are used for arrival rates.

Description: The Exponential distribution is a continuous distribution that is bounded on the lower side. Its shape is always the same, starting at a finite value at the minimum and continuously decreasing at larger x. An Exponential distribution does not have a standard deviation; the distribution is defined by the mean value. The most frequently occurring values are close to zero. As the value increases, the number of occurrences decreases.

ExtremeValueTypeA Distribution

Syntax: Distributions.ExtremeValueTypeA(*mean*, *standard deviation*)

Applications The Extreme Value Type A distribution describes the limiting distribution of the extreme values of many types of samples. Examples include parameters in growth models, astronomy, human lifetimes, radioactive emissions, strength of materials, flood analysis, seismic analysis, and rainfall analysis.

Description: The Extreme Value Type A distribution is a continuous distribution that is unbounded. Its shape is always the same but it can be shifted or scaled to need. It is also called the Gumbel distribution.

ExtremeValueTypeB Distribution

Syntax: Distributions.ExtremeValueTypeB(*mean*, *standard deviation*)

Applications: The Extreme Value distribution describes the limiting distribution of the extreme values of many types of samples. The Extreme Value Type B describes the limiting distribution for the *lowest* value samples whereas the Extreme Value Type A distribution describes the limiting distribution for the *highest* value samples. The Extreme Value Type B distribution is often used to represent parameters in growth models, astronomy, human lifetimes, radioactive emissions, strength of materials, flood analysis, seismic analysis, and rainfall analysis.

Description: The Extreme Value Type B distribution is a continuous distribution that is unbounded. Its shape is always the same but it can be shifted or scaled to need. It is also called the Gumbel distribution.

Gamma Distribution

Syntax: Distributions.Gamma(mean, standard deviation)

Gamma(shape)

Applications: The Gamma distribution is often used to model nonnegative random variables, such as the time for a human operator to complete a task. The Gamma distribution is also used to represent lifetimes, lead times, personal income data, a population about a stable equilibrium, and service times. Use the Gamma distribution for task times when a task cannot be done much faster than the mean time, yet could take much longer.

Description: The Gamma distribution is a continuous distribution that is bounded at the lower side. It has three distinct regions. The range of values above the mean is much greater than the range of values below the mean.

Geometric Distribution

Syntax: Distributions.Geometric(mean)

Applications: The Geometric distribution is used to represent the number of failures before the first success in a sequence of independent Bernoulli trials. The Geometric distribution can be used to represent the number of items inspected before encountering the first defective item, the number of items in a batch of random size, or the number of items demanded from an inventory.

Description: The Geometric distribution is a discrete distribution with a lower bound at 0 and no bound on the high side.

InverseGaussian Distribution

Syntax: Distributions.InverseGaussian(mean, standard deviation)

Applications: The Inverse Gaussian distribution is used to model Brownian motion and diffusion processes with boundary conditions and to model the distribution of particle size in aggregates, reliability and lifetimes, and repair time.

Description: The Inverse Gaussian distribution is a continuous distribution that is bounded on the lower side. It is uniquely zero at the minimum x and always positively skewed. The inverse Gaussian is also called the Inverse Normal Distribution or Wald Distribution.

Logistic Distribution

Syntax: Distributions.Logistic(mean, standard deviation)

Applications: The Logistic distribution is most often used as a growth model for populations, for weight gain, and for business failure. Occasionally, the Logistic distribution is used in place of the Normal distribution where exceptional cases play a larger role.

Description: The Logistic distribution is a continuous distribution that is unbounded and symmetrical about its mean. The shape of the Logistic distribution is very much like the Normal distribution, except that the Logistic distribution has broader tails.

LogLogistic Distribution

Syntax: Distributions.LogLogistic(scale, shape)

Applications: The LogLogistic distribution is used to model the output of complex processes such as business failure and product cycle times.

Description: The LogLogistic is a continuous distribution that is bounded on the lower side. A true Loglogistic distribution occurs when the shape parameter is set to 1.

Lognormal Distribution

Syntax: Distributions.Lognormal(mean, standard deviation)

Applications: The Lognormal distribution describes values with normally distributed natural logarithms. It is often used to describe random processes that represent the product (rather than the sum) of several small, independent events. The Lognormal distribution is used in many different areas, including the distribution of particle size in naturally occurring aggregates, dust concentration in industrial atmospheres, duration of sickness absence, lifetime distributions in reliability, distribution of income, employee retention, and many applications modeling weight, and height.

Use the Lognormal distribution for tasks that cannot be done much faster than the mean but sometimes take much longer. Because the Lognormal distribution generally has a longer tail than the Gamma distribution, it may be more appropriate for tasks with no known upper bound on their time duration in extreme situations.

Description: The Lognormal distribution is a continuous distribution that is bounded on the lower side. It is always 0 at minimum x, rising to a peak that depends on both the mean and standard deviation, and then decreases monotonically for increasing x.

NegativeBinomial Distribution

Syntax: Distributions.NegativeBinomial(successes, probsuccess)

Applications: The Negative Binomial distribution provides a good approximation for the sum or mixing of other discrete distributions. By itself, it is used to model accident statistics, birth-and-death processes, market research and consumer expenditure, lending library data, and biometrical data.

Description: The Negative Binomial distribution is a discrete distribution that is bounded on the low side at 0 and unbounded on the high side.

Normal Distribution

Syntax: Distributions.Normal(mean, standard deviation)

Applications: The Normal distribution is commonly used for errors of various types, such as in the impact point of a bomb or quantities that are the sum of a large number of other quantities.

Description: The Normal distribution is a continuous distribution that is unbounded. It is also known as the bell curve. In a Normal distribution, each value is equally likely to fall on either side of the mean. A plot of the frequency for the occurrence of each value is bell-shaped, with the highest point of the bell occurring at the mean value. The Normal distribution is frequently used to represent symmetrical data, but is unbound in both directions. If the data are known to have a lower bound, it may be better represented by suitable parameterization of the Lognormal, Weibull, or Gamma distributions. If the data are known to have both upper and lower bounds, the Beta distribution can be used.

Pareto Distribution

Syntax: Distributions.Pareto(minimum, scale)

Applications: The Pareto distribution has historically been used to represent the income distribution of a society. It is also used to model many empirical phenomena with very long right tails, such as city population sizes, occurrence of natural resources, stock price fluctuations, size of firms, brightness of comets, and error clustering in communication circuits.

Description: The Pareto distribution is a continuous distribution that is bounded on the lower side. It has a finite value at the minimum x and decreases monotonically for increasing x.

PearsonType5 Distribution

Syntax: Distributions.PearsonType5(scale, shape)

Applications: The Pearson Type 5 distribution is useful for modeling time delays where some minimum delay value is almost assured and the maximum time is unbounded and variably long. Examples include the time to complete a difficult task, time to respond to an emergency, and time to repair a tool, and so on.

Description: The Pearson Type 5 distribution is a continuous distribution that is bounded on the lower side. The Pearson Type 5 distribution is sometimes called the Inverse Gamma distribution due to the reciprocal relationship between a Pearson Type 5 random variable and a Gamma random variable.

The Pearson Type 5 distribution starts slowly near its minimum and has a peak slightly removed from it. With decreasing shape parameter, the peak gets flatter and the tail gets much broader.

Pearson Type 6 Distribution

Syntax: Distributions.PearsonType6(scale, shape1, shape2)

Applications: The Pearson Type 6 distribution is rarely used, except in its reduced form as the F distribution. In this case, the distribution is used in the analysis of variance and is a macro of the ratio of two independent random variables each of which has a chi-square distribution and is divided by its number of degrees of freedom.

Description: The Pearson Type 6 distribution is a continuous distribution that is bounded on the low side. It has three distinct regions dependent on the scale and shape factors.

Poisson Distribution

Syntax: Distributions.Poisson(mean number of occurrences per interval)

Applications: The Poisson distribution models the number of independent events that occur in an interval of time. This includes many types of events in time or space, such as arrivals of telephone calls, defects in semiconductors manufacturing, defects in all aspects of quality control, molecular distributions, stellar distributions, geographical distributions of plants, shot noise, and so on. Note that the time between arrivals (defects) is Exponentially distributed, which makes this distribution a particularly convenient starting point even when the process is more complex.

Description: The Poisson distribution is a discrete distribution that is bounded at 0 on the low side and unbounded on the high side.

Rectangular (Uniform) Distribution

Syntax: Distributions.Rectangular(mean, minimum)

Applications: The Rectangular distribution is used to represent a random variable with constant likelihood of being in any small interval between minimum and maximum and centered around the mean. The Rectangular distribution is primarily used to represent mechanical tasks rather than natural processes.

For example, when a pedestrian encounters a Don’t Walk light at an intersection, the amount of time she has to wait for a Walk light is rectangularly distributed. The wait could take zero seconds or as long as the maximum length of the Don’t Walk light.

Description: The Rectangular distribution is a continuous distribution bounded on both sides. It is a special case of the Beta distribution. It is also called the Uniform distribution. Most random number generators provide samples from the Uniform distribution on (0,1) and then convert these samples to random variates from other distributions.

Triangular Distribution

Syntax: Distributions.Triangular(mode, minimum, maximum)

Applications The Triangular distribution is often used when no or little data are available; it is rarely an accurate representation of a data set. However, it is employed as the macro form of regions for fuzzy logic due to its ease of use.

Description: The Triangular distribution is a continuous distribution that is bounded on both sides. The Triangular distribution can take on very skewed forms including negative skewness. For the exceptional cases where the mode is either the minimum or maximum, the Triangular distribution becomes a right triangle.

Weibull Distribution

Syntax: Distributions.Weibull(scale, shape)

Applications: The Weibull distribution is often used to model the strength of materials or the mean time to failure for a device. Industrial applications of survival analysis often involve testing components to destruction after subjecting them to a stress which is assumed to speed up the aging process. In particular, the Weibull distribution is used to represent wear-out lifetimes in reliability, duration of industrial stoppages, and thunderstorm data.

Description: The Weibull distribution is a continuous distribution that is bounded on the lower side.

Glossary

Accuracy Measure

Accuracy Measure is a unit corresponding to the nature of how your task's accuracy is measured. This unit helps IMPRINT Pro to determine whether your task is more or less accurate when the number pulled from the accuracy distribution is greater or less than the requirement you set. For example, assume that your Accuracy Measure is set to Percent Steps Correct and your Accuracy Requirement is set to 80. If the number pulled from the Accuracy distribution is 85, then IMPRINT Pro will consider this accuracy value to be a success since 85% is better than 80% in the context of Percentage of Steps Correct. If, however, you intend choose the unit of Measure Feet from Desired, IMPRINT Pro will consider this accuracy value of 85 to be a failure since 85 is less accurate that 80 in the context of Feet from Desired.

Accuracy Requirement

The Accuracy Requirement is the minimum acceptable accuracy for a task in order for it to be considered a success.

Accuracy Standard Deviation

The Accuracy Standard Deviation specifies how tightly all the various values in the distribution are clustered around the Mean Accuracy value.

Activities Trump Matrix

A matrix used to set task priority within a Force Unit schedule for when any two activities overlap.

Analysis Description

A brief description of the analysis found in the Properties window. The Analysis Description is optional.

Analysis Name

The Analysis Name can be 20 characters long and can include spaces and symbols. When an analysis is added, IMPRINT Pro automatically generates an analysis name that the user may edit in the Properties window. However, when an analysis is saved, the user will need to enter a name in the dialog provided.

Analysis Version

The Analysis Version can be 20 characters long and can include spaces and symbols. When an analysis is added, IMPRINT Pro automatically generates an analysis version that the user may edit in the Properties window. When an analysis is saved, the user will need to enter a version number in the dialog provided.

Array

An ordered set of variable values that are indexed to a single variable name. An array can be a one-dimensional list, a two-dimensional table of rows and columns, or a multi-dimensional array.

Beginning Effect

An expression used to change the values of variables in the model as a result of a task starting.

Clock

System variable that records elapsed time in simulation time units since the beginning of model execution. The Clock variable can be used in any expression in a model.

Contingency Operators

One or more contingency operators may be selected for each task. During the task reallocation process, a contingency operators will be considered if workload overload occurs during mission execution.

Crew Ratio

Crew Ratio is the average number of crews available for each new system in the unit. In some cases where systems are required to operate continuously there may be multiple crews for each system.

Criterion

This value is the percentage of time that this task, function or mission must meet both its time requirement and its accuracy requirement simultaneously in order to be considered successful.

Decision Node

A diamond-shaped object on a network diagram containing one of the following letters: P, M, or T—representing a Probabilistic, Multiple, or Tactical decision.

Decision Type

The Decision Type determines which path(s) an entity may follow when exiting a node which branches out along multiple paths. Tasks with a decision type of Multiple will follow all paths every time; tasks with a decision type of Probabilistic are likely to follow the path(s) in proportion to the percentages you set for each path; tasks with a decision type of Tactical will follow only those paths whose conditions evaluate to True; tasks with a decision type of Single will only ever follow one path.

Distribution

The range of all possible values which can be assigned to a task's time value and the probability that those values will be chosen at random.

Effect

An expression that executes as a result of task activity during model execution. Tasks can have beginning, and ending effects. You can use these effects to change variable values and thus represent the system changes that occur as a result of the task activity.

Ending Effect

An expression used to change the values of variables in the model because of a task ending.

Entity

A conceptual object that travels through a task network and indicates by its location when each task is executing or waiting to execute. The entity may represent a physical object, such as a part being built on a production line. It may also represent a person performing the tasks. Or, the entity may simply trace the sequence of tasks and not correspond to anything physical. During model execution, entities can be represented by symbols or numbers that travel through the network diagram. Entities are identified by corresponding values of its entity attributes.

Entity.Duration

Entity attribute that records the time each entity spends in the current task.

Entity.Tag

Entity attribute that records the identity of each entity when there are multiple entities traveling through a network. Once an entity has a Entity.Tag value, the value stays with the entity through the remainder of model execution.

Estimated Task Accuracy

Use Task Mean Accuracy and Accuracy Standard Deviation under this section of the TimeAcc tab to specify an average value for how accurately a particular task is likely to be performed. For example, you might say that on the average when performing this task the operators get 90 percent of the steps correct or for another tasks they are within 10 mils of the correct azimuth. Under the first option, you will also specify the standard deviation, which in some sense is a measure of the worst and best the task is likely to be performed. An easy rule of thumb for specifying the standard deviation is to compute the difference between the worst and best performance and divide it by 6. For example, if the worst performance is 40% steps correct and the best is 100%, then an estimate for the standard deviation would be 10% (((100 - 40)/6) = 10). Under this option you will define what is acceptable performance by specifying an accuracy standard. For example, you may say that any error greater 5 mils causes firing inaccuracies that are unacceptable. With the information above, the model then computes the probability of the task performance being at or above the standard and compares a selected random number to determine if task was performed accurately.

Estimated Task Time

The Estimated Task Time is the amount of time a task will take to execute as calculated by IMPRINT Pro. This time will vary each time the task is executed. To calculate this time, IMPRINT Pro starts with an initial task time which can either be represented as a value (entered in the Value field in the format of HH:MM:SS.mm) or as a time expression (entered in the Expression field and evaluated to seconds). Next, IMPRINT Pro pairs this initial time with a statistical distribution type. This distribution determines the possible values your task time could take relative to your initial value. When IMPRINT Pro runs, a random number is generated and is used to pull a number from the distribution you have specified. This number becomes the Estimated Task Time for that occurrence of the task. Note: of the available 20 distribution types, IMPRINT Pro uses, by default, the Normal distribution which uses Mean and Standard Deviation to predict a task performance time. Depending on the distribution type you choose to use, different and/or additional parameters such as Shape, Probability of Success, Maximum, Minimum and Scale, might also be required.

Event

Something that is scheduled to happen during a simulation at a specific clock time. Some events are expressions that you schedule to occur at specific times—these are called scenario events. All other events are scheduled, such as the finish times for currently executing tasks, as the model is running. You can watch events being scheduled by displaying the Event Queue window.

Event Queue

Events that are scheduled to happen during model execution are placed in an Event Queue. As each event occurs, it is removed from the top of the queue. These events are displayed in the Event Queue window.

Expression

A calculation, formula, macro, or statement that supplies a value or performs an operation. Expressions can contain combinations of numerical values (constants or variables), macros, mathematical and logical operators, and logical statements.

Force Unit

A Force Unit is a group of individuals, defined by jobs, who perform a variety of activities according to schedules.

Function

A network that is inside another network. In a model, all networks are functions except for the top network (level 1). You can create a function by clicking on your network diagram with the Network tool, then selecting the function with the pointer and opening its diagram. You draw the function diagram in the same way as you draw the main network diagram.

Function Criterion

Function Criterion is a percentage that determines how often the function must meet its Function Time Requirement to be considered a success. Enter a value between 0 and 100, inclusive.

Function Time Requirement

The Time Requirement is the slowest performance time that can be tolerated and have the function still be considered a success. When IMPRINT Pro executes your mission model, the performance time that is predicted by the aggregation of all tasks in this function will be compared to this standard to ensure that your design can meet the function level time requirement. The format for this entry (as with all other time values) is HH:MM:SS.mm. To enter a time requirement of 30 minutes, 15 seconds enter 00:30:15.00. Notice that you can enter hundredths of a second after the decimal point.

Job

The occupation held by one or more members of the Force Unit. Each job is defined by a name, specialty, rank and role. Examples may include Tank Driver, Navigator and Analyst.

Length of run

The number of days of operations that the IMPRINT Pro maintenance model will simulate. If you define mission segments for more days than are entered here, the model will not execute any segments that go beyond this value. This value defaults to 1.00.

Macro

One or more mathematical or logical expressions that are assigned a single name and return a single value. When a macro name is encountered in an expression in a model, the model executes the expressions included in the macro and returns a value. IMPRINT Pro contains model, mathematical, distribution, and Animator macros for your use or you can define and save your own macros with the model.

Maximum Number of Systems

This value is the maximum number of systems that would be assigned to your segment if available.

Mean Accuracy

The Mean Accuracy is the estimate of the most likely, or average, accuracy value for a task.

Mean Time Expression

The Mean Time represents the average number of seconds required to execute a task. You can either enter a mean time value (00:00:30) or you can enter an expressionclock<=30 then 10 else 15). Each expression is delimited by semicolons. In building the expression you can use any of the algebraic or logical operators including the following: (&, |, >, <, :=,==, +, -,*, and /), and you can use if-then-else statements. The assignment operator is :=. The equivalence operator is ==.

Minimum Number of Systems

This value is the minimum number of systems that must be ready to begin the segment prior to the cancellation time in order to prevent all departure groups from being canceled.

Mission Accuracy Criterion

The Mission Accuracy Criterion is a percentage that determines how often the mission must complete without abort to be considered a success. Aborts are caused when a task fails and the consequence of failure is mission abort Values can range from 0 to 100.

Mission Criterion

The Mission Criterion is the percentage that represents how often the mission must meet both its time and accuracy standards at the same time. Values can range from 0 to 100.

Mission Description

The Mission Description is a fairly long text field that you can use to describe the mission you intend to model. The description is limited to 255 characters. All printable text characters are allowed.

Mission Name

The Mission Name is a brief text label that you will use to uniquely identify your system's mission. The mission name is limited to 60 characters. All printable text characters are allowed, including spaces.

Mission Time Criterion

The Mission Time Criterion is a percentage that determines how often the mission must meet its Mission Time Requirement to be considered a success. Values can range from 0 to 100.

Mission Time Requirement

The Time Requirement is the slowest performance time that can be tolerated and have the mission still be considered a success. When IMPRINT Pro executes your mission model, the performance time that is predicted by aggregating your individual tasks will be compared to this standard to ensure that your design can meet the mission level time standard. The format for this entry (as with all other time values) is HH:MM:SS.mm. So, to enter a time requirement of 30 minutes, 15 seconds enter 00:30:15.00. Notice that you can enter hundredths of a second after the decimal point.

MMH/System

Annual MMHs/system (ORG, DS, & GS levels) are the total number of annual maintenance manhour (by maintenance level) required to maintain one new system in that unit.

New Systems

This is the number of new systems that will be fielded in the unit.

Network

A sequential relationship of jobs (tasks and functions) that simulate a system, activity, or process. You create the network by drawing a network diagram, and you use variables to simulate the effects of the tasks on other tasks and on the system.

Network Diagram

A graphical depiction of a model network showing the sequence of tasks and functions, and the possible paths through the network. You draw the network diagram using the tools in the network diagram palette.

Number of times to run the mission

If you would like to run a model more than once, you can enter that number in this field. If you have a relatively short scenario (in days), you may want to run the model several times in order to be confident that the data you get from all of the runs is representative. One short run may, by chance, produce data that are not very typical. Running a model for a long period of time (for example, 90 days) will also improve the probability that the results are typical. IMPRINT Pro will automatically begin each run with a new random number seed. This value defaults to 1.

Number Per Departure Group

This value controls the number of systems per group (sometimes referred to as a “flag”) that are sent out on this segment.

OP Tempo

Annual OP Tempo is the usage per year that the new system is expected to accrue in the type and size unit listed.

Overall Workload

The sum of all single task demand values and all conflict values (inter-channel and intra-channel) of all ongoing tasks that an operator is currently performing at a given instant of time (see Operator Workload Summary and Detail reports.)

Path

A sequential connection between two nodes (tasks or functions), drawn by dragging the mouse cursor from one node (source) to another (destination). The conditions under which the path is taken is specified in the Paths tab of the source node.

Planned Activity

A planned activity is an activity comprising part of a Force Unit schedule, which has a designated start time and will last until the next scheduled activity is set to occur. Examples may include “guard duty,”” hygiene,” “eating” and “sleeping.”

Primary Operator

The Primary Operator is the Specialty and crew position that will be assigned to perform the task.

Probability of Success

The Probability of Success is the likelihood that a task will succeed each time it is executed. This number can either be manually entered or can be calculated as a result of entering values for the remaining three fields upon which it is dependent, namely the Mean Accuracy, Accuracy Standard Deviation and the Accuracy Requirement.

Random Number Seed

A number between approximately -2 billion and +2 billion used to generate random numbers for calculating task execution times and taking probabilistic paths through the network. You supply the random number seed when you define the Execution settings for a model.

Rank

An official position or standing in the armed forces. This information is used to help define the jobs in a force unit. Ranks include E1-E3 (Private and Private First Class), E4 (Specialist or Corporal), E5 (Sergeant), E6 (Staff Sergeant), E7 (Sergeant First Class), and E8-E9 (Master Sergeant, First Sergeant, Sergeant Major, Command Sergeant Major and Sergeant Major of the Army).

Release Condition

An expression that is evaluated to determine whether or not a task can execute. A task can execute only when the value of the release condition is nonzero or true. You supply the release condition when you define the task. The default release condition is always “true”.

Repeating Segments

A Repeating Segment is one in which IMPRINT Pro’s maintenance model will attempt to send more systems out to perform this segment every so often. You can identify the frequency of the segment (for example, “repeats every four hours”).

Replacement Systems

This is the number of old systems that will be replaced in a unit.

Resource-Interface Channels

Resource-Interface Channels are formed by pairing the human resources with the interfaces (for example, visual/heads up display, motor/control stick). All workload is assigned to a task by setting individual single task demand values for each resource-interface channel used to perform that task.

Role

The level of participation that a job (an occupation held by one or more members of the Force Unit, such as a driver or gunner) will have in an unplanned activity if one requiring such a level occurs. Roles include Leader, Sub-Leader and Member.

Run

The current run number in cases where there are multiple runs. The run number can be accessed through the expression `Model.RunNumber`.

Scenario Description

The scenario description is a free text field that you can use to document your analysis. It is limited to 255 characters.

Scenario Name

This is the name of your maintenance scenario. You can have many maintenance scenarios for each system.

Seed

A random number seed used to run a model. The random number seed generates a set of random numbers that are used to calculate task execution times and probabilistic paths taken through the network. The random number seed can be set in the Execution dialog box. For multiple runs of a model, Model.RandomSeed contains the random seed used to generate the particular run.

Segment Cancellation Time

This is the number of hours that the simulation will wait for the minimum number of systems to be available before canceling the segment. For example, if the minimum number of systems for this segment is six and there are only four available at the scheduled start of the segment then the segment must wait. If there is a value of “01:00:00.00” in this field, then if the minimum number of systems do not become available within one hour of the scheduled start, the segment will be canceled. The default value for this item is 00:00:00.00. This means that the minimum number of systems must be available at the start time of this segment in order for the segment to begin. Be very careful that your cancellation is not after the segment would be attempting to repeat.

Segment Duration Time

This value indicates the number of operational hours for this segment. This value is used to schedule the return of the systems from operations and the beginning of the maintenance window. Additionally, this time is used to determine usage for the equipment on which failures are triggered by time.

Segment Priority

When more than one segment has been defined to occur at the same time, the segments must compete for available systems. In this box, you can enter a priority number for each segment. Available systems will be allocated to higher priority segments first.

Segment Repeat Mean Time

If you have checked the Repeating box for your segment, you will be able to enter a time in the Segment Repeat Time box. The value in this box controls the frequency (in hours) with which the simulation will attempt to repeat this segment. If you will enter the number 4 in this box, it will indicate that you want this segment to repeat every four hours. In other words, this segment will try to start again every four hours throughout the simulation. If you wanted this segment to repeat only twice, you could do this by defining two separate but identical segments with one starting four hours after the first.

Segment Start Day

The Segment Start Day and the Segment Start time work together to identify the start time of this segment. This value defaults to 1. If you were to leave the default values in both of these boxes, this mission would begin at time 0 on day 1.

Segment Start Time

The time entered in the Segment Start Time box indicates the time the segment is to begin. This value defaults to 00:00:00.00.

Shift Data

Enter the maintenance shift length in clock hours and the number of shifts per day. The product of the two numbers must be less than or equal to 24. The number of shifts will determine the number of shift columns in the Maintenance Crew spreadsheet.

Shift Length

The length of shift in hours.

Shifts per day

The number of shifts per day.

Snapshot

An option that records the values of specified variables at particular points during model execution—for example, when a task starts or ends, at the end of a run, or at specific clock times. When you run a model with snapshots turned On, the values are recorded, and results may be viewed in the Snapshots report.

Specialty

A specialty is a designation for a particular military job within the Armed Forces referenced by a 3-letter designation. For example, a Scout helicopter repairer is designated 67S. This option enables you to select people of appropriate specialties for task assignments.

Spinner Task

A task with a path that goes back to itself. Spinner tasks are used to generate multiple entities to travel through the network.

Subsystem Equipment Group Name

The subsystem type indicates the operational units (Armament, Mobility, and Other) that are used to describe the usage for that subsystem.

Subsystem

A subsystem is one of several smaller elements comprising a system, for example, an engine is a subsystem that is part of a system called “Tank”. Each Subsystem is going to be one of three types (Armament, Mobility, and Other) and must be uniquely named. Subsystem type indicates the operational units that are used to accrue the usage for that subsystem Usage for an Armament subsystem is described as the number of rounds that have been fired from that subsystem. Usage for components in a Mobility system is described in terms of the distance that the system has traveled. Subsystems with a type of “Other” use operational time as the usage measure.

Task

Tasks are the basic building blocks of a model. A task is defined by timing information, execution constraints (or release conditions), effects of the task on the system, and routing information concerning following tasks and functions. Each time a task executes, the execution time is randomly calculated within the time distribution parameters you supply. You can create a task using the Palette and Network Diagram or through the Analysis Tree. To define a task simply double-click the task in the tree or in the network diagram, and enter its information in the Task Properties window.

Task Accuracy Probability

As an alternative to specifying a task’s Accuracy Mean, Standard Deviation and Accuracy Standard data and relying upon IMPRINT Pro to calculate its Probability of Success, you can instead directly enter this value by (as a percentage) in the Probability of Success field. When the task runs, the value entered here is compared with a random number to determine if the task was performed accurately.

Task Criterion

The percentage of time that the task must meet its time and accuracy standards in the same simulated occurrence in order to considered a success.

Task Mean Accuracy

The average value for how accurately a particular task is likely to be performed.

Task Name

This text box displays the name of the task for which you are displaying or entering information.

Task Priority

On the Effects tab, task priorities are used to execute priority-based workload management strategies.The priorities range from 1 (very low) to 5 (very high). New tasks are assigned a priority level of 3 (medium). To change a priority level for a task, double-click on a task to open the task parameter window, click in the Task priority field of the task and enter a priority value between 1 and 5, or click on the drop box in the task priority field and select a priority from the list.

Task Standard Deviation

A measure that tells you how closely all the various task times calculated by IMPRINT Pro are clustered around the mean task time. In general, one standard deviation away from the task mean time in either direction (greater than or less than the mean) accounts for somewhere around 68 percent of all computed task times. Two standard deviations away from the mean account for roughly 95 percent of the computed task times. And three standard deviations account for about 99 percent of the computed task times. By specifying the Task Standard Deviation for your task, you are helping IMPRINT Pro to determine how widely varied (or unvaried) the distribution of all your different calculated task times will be.

Task Time Requirement

The Time Requirement is the slowest performance time that can be tolerated and have the task still be considered a success. When IMPRINT Pro executes your mission model, the Distribution along with parameters Mean and Standard Deviation are used to choose a specific task time for each occurrence of each task. These times are then accumulated throughout the model run and compared to the time standard in order to report the percentage of successes for each task. To enter the time requirement, use the mouse to position the cursor in the time requirement box.

Time Between Departures

The value in the Time Between Departures field denotes the amount of time that must lapse between any two consecutive group departures for a given segment. This value is in hours and can be entered in the HH:MM:SS.mm format to include hundredths of hours.

Time Format

Time data entered in the IMPRINT Pro interfaces or viewed in the IMPRINT Pro reports can be displayed in one of several different formats, such as hours, minutes, seconds and HH:MM:SS.mm. Time formats may be changed in the Time tab through the Preferences option under the View menu.

Time Requirement

The Time Requirement is the maximum acceptable performance time for a task.

Unplanned Activity

A unplanned activity is an activity which can interrupt the normal planned activities comprising a Force Unit schedule, Unplanned activities are defined by a probable start time, duration and the number of people required to address them. Examples may include “fire” and “emergency”.

Variable

An identifier which is used to represent a quantity which can vary in value. When a variable name is encountered in an expression, IMPRINT Pro substitutes in the current value of the variable and evaluates the expression. Variables are useful for representing changeable aspects of the system you are simulating. You can change the value of a variable through any expression in your model, for example, in a task effect or in a scenario event. For each IMPRINT Pro model, five system variables are automatically created including Clock, Distributions, Entity, Model, and Task. For more information about system variables see the “Variables” section in this manual.

Warfighter

A warfighter is any person or automated device that operates, maintains, supplies or supports military equipment.

Appendix A: Check for Errors Option

Before running an operations model in IMPRINT Pro, use the **Check for Errors** option located under the **Execution** menu to detect any syntax errors that may be in any task or macro fields containing expressions. These fields include but are not limited to the **Release Condition**, **Beginning Effect**, **Ending Effect, Mean Time Expression** and **Paths** fields and the **Macro** expression field.

IMPRINT Pro generally indicates the nature of the problem in the error message, such as “unrecognized word” for a misspelled or undeclared variable or function, or “semicolon expected at end” for a missing semi-colon.

Common problems include typographical errors in an expression, omitting a semi-colon, forgetting to define a variable, or using the wrong case for a variable name. Because IMPRINT Pro distinguishes between upper and lower case letters, name, NAME, and Name are considered to be different and unique variables.

Errors may occur as a result of the conversion of IMPRINT 7 models into IMPRINT Pro where syntax methods differ.

Using Check for Errors

To check for syntax errors:

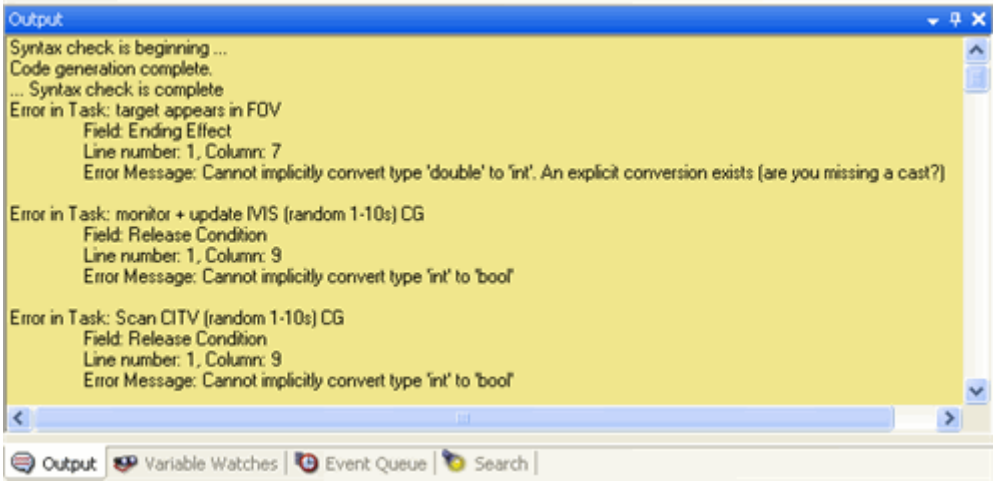
1. From the **Execution** menu, click **Check for Errors**, or use shortcut keys **Ctrl+Shift+G**.

   Any errors that are detected display in the **Output** window.

2. To open the dialog box where the error occurs, double-click the error.

Error Displays

Errors display in the output window as follows:



- **Error in Task/Macro.** The name of the task or macro whose syntax is in question.
- **Location of the error**. The line number and column number inside the code window where the error may be found.
- **Type/Field.** Errors which occur in any one of the code boxes in the Effects tab of a task will be designated as a **Type** error, and specifically as one of the following types: Release Condition, Beginning Effect or Ending Effect. For example an incorrect operator expression in a task's Release Condition field would result in an error of type Release Condition.

  Errors occurring in any other field in the task tabs are designated as a **Field** error, for example a missing semicolon in a Mean Time Expression on the Time and Accuracy tab.

  No Type or Field comments appear for macros containing syntax errors.
- **Line number and column number.** The line number inside the code window where the error can be found (line numbers appear to the left hand side of code in blue.)
- **Error message.** Specific details about the syntax in question.

To correct syntax errors which appear in the **Output** window, double-click any one of the four lines above – the IMPRINT Pro document window immediately updates to show you to the location of the syntax error in your model. The text in question appears underlined in red. Use the guide below to for examples of commonly-found errors and the syntax changes required to correct them.

Sample Errors and Corrections

Sample Error 1:

Error in Task: monitor + update IVIS (random 1-10s) CG
Type: ReleaseCondition
Line number: 1, Column: 9
Error Message: Cannot implicitly convert type 'int' to 'bool'

Problem: The Return statement is trying to return a true or false value. The variable by itself is insufficient:

Release Condition:

```
1 return( Cbusy);
```

Solution: Make the variable part of an expression that evaluates to true or false.

Release Condition:

```
1 return( Cbusy==1);
```

Sample Error 2:

Error in Task: Rejoin tasks 5-7
Type: ReleaseCondition
Line number: 1, Column: 9
Error Message: Operator '&&' cannot be applied to operands of type 'int' and 'int'

Problem: The return statement is trying to return a true or false value. The variables by themselves are insufficient:

Release Condition:

```
1 return( Path5 && Path6);
```

Solution: make the variable part of an expression.

Release Condition:

```
1 return( Path5==1 && Path6==1);
```

Sample Error 3:

Error in Task: CG prepares to reload (2nd rd on) no workload autoloader
Type: TimeMean
Line number: 9, Column: 8
Error Message: Invalid expression term '}'

Problem: a return statement is missing after the “if” statement, and the final return statement is empty.

```
1 if (Loaded==0)
2 {
3       0;
4
5 }
6 else
7 {
8 5
9 return());
```

Solution: Add complete return statements after the “if “statement and the “else” statement.

```
1 if (Loaded==0)
2 {
3     return 0.0;
4
5 }
6 else
7 {
8     return 5.0;
9 }
```

Alternative fix: add an expression after the “if” statement.

```
1 double meantime=0.0;
2 if (Loaded!=0)
3 {
4     meantime=5.0;
5 }
6     return meantime;
7
```

Sample Error 4:

Error in Task: CG prepares to reload (2nd rd on) no workload autoloader
Type: TimeMean
Line number: 9, Column: 2
Error Message: } expected

See correction in error message above.

Sample Error 5:

Error in Task: CG prepares to reload (2nd rd on) no workload autoloader
Type: TimeMean
Line number: 8, Column: 3
Error Message: ; expected

See correction in error message above.

Sample Error 6:

Error in Task: Avoid Obstacle or Redirected Driver
Type: BeginningEffect
Line number: 6, Column: 7

Error Message: Cannot implicitly convert type 'double' to 'int'. An explicit conversion exists (are you missing a cast?)

Problem: the current variable's type does not support code in the Effects or Macros window.

Beginning Effect:

```
3 Hinderance=0;
4 Prob4=Model.Random();
5 if (Prob4<0.10)
6 {
7         Tgtappear=1;
8 }
```

Mission Name: Commander-gunner + Driver Treatment 3
Variable Name: Prob4
Variable Type: Integer
Is Array Dimensions:
Initial Value: 0
Cultural: false

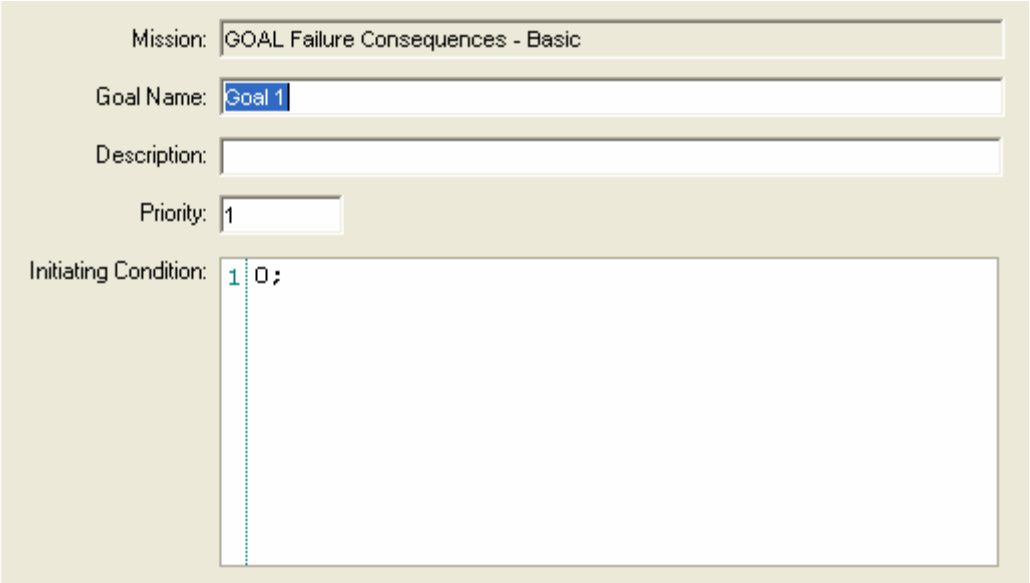
Solution: change the variable type for the given variable.

Mission Name: Commander-gunner + Driver Treatment 3
Variable Name: Prob4
Variable Type: FloatingPoint
Is Array Dimensions:
Initial Value: 0.0
Cultural: false

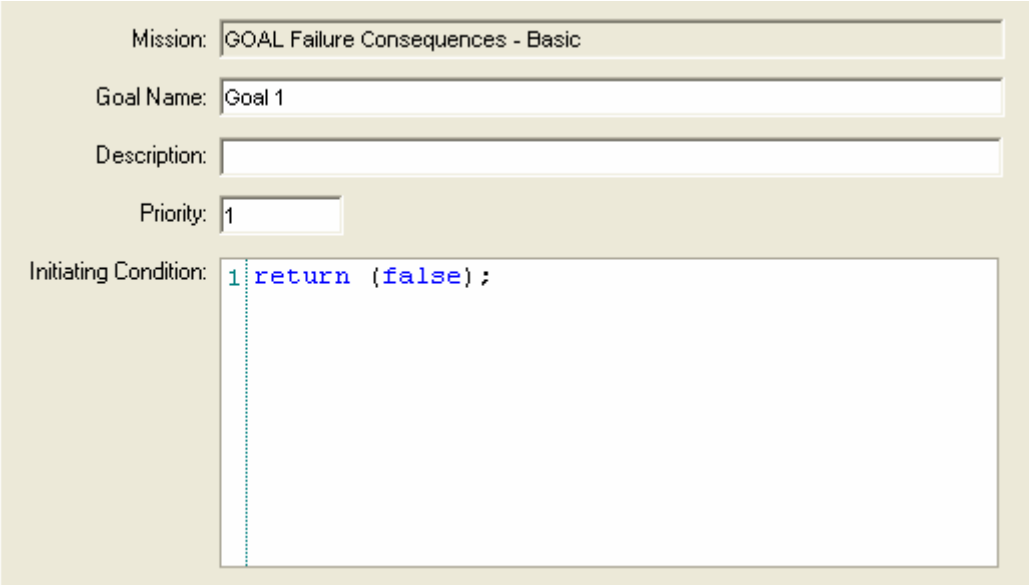
Sample Error 7:

Error in Goal: Goal 1
Type: InitiatingCondition
Line number: 1, Column: 1
Error Message: Only assignment, call, increment, decrement, and new object expressions can be used as a statement

Problem: The initiating condition of a goal is incorrectly set.



Solution: Change the initiating condition of the goal to true or false using a return statement.



Sample Error 8:

Error in Task: Task 6-Rejoin Node
Type: DecisionCode
Line number: 1, Column: 1
Error Message: Only assignment, call, increment, decrement, and new object expressions can be used as a statement

Problem: Decision code is not in the form of a return statement.

Decision Code:

```
1  counter==20;
```

Solution: Add a return statement to the Decision code.

Decision Code:

```
1  return (counter==20);
```

Sample Error 9:

Error in Task: Task 9-set counter to 1
Type: EndingEffect
Line number: 3, Column: 9
Error Message: Invalid expression term '='

Problem: The assignment operator for the variable is incorrect (the convention ":=" is the assignment operator used by previous versions of IMPRINT.)

Ending Effect:

```
1 counter:=1;
```

Solution: change the assignment operator to "=".

Ending Effect:

```
1 counter=1;
```

Index

A

Abort macro 27
Abort(array) macro 28
Abs macro 41
accuracy measure, definition 71
accuracy requirement, definition 71
accuracy standard deviation, definition 71
Acos macro 41
activities trump matrix, definition 71
adjustment operators 15
analysis description, definition 71
analysis name, definition 71
analysis version, definition 72
arithmetic operators 14
array, definition 72
Asin macro 41
assignment operator 15
Atan macro 41
Atan2 macro 42

B

Beep macro 28
beginning effect, definition 72
Bernoulli distribution 57
Bernoulli macro 48
Beta distribution 58
Beta macro 48
BigMul macro 42
Binomial distribution 58
Binomial macro 49

C

Ceiling macro 42
clock, definition 72
commas in expressions 12
comments in expressions 12
constants
    defined 9
contingency operators, definition 72
continuous probability distributions 55
Cos macro 42
Cosh macro 43
CreateEntity macro 28
crew ratio, definition 72
criterion, definition 72

D

decision type, definition 73
Direction macro 29
discrete probability distributions 55
DiscreteUniform distribution 59
DiscreteUniform macro 49
Distance macro 29
distribution macros
    Bernoulli 48
    Beta 48
    Binomial 49
    DiscreteUniform 49
    Exponential 49
    ExtremeValueTypeA 50
    ExtremeValueTypeB 50
    Gamma 50
    Geometric 51
    InverseGaussian 51
    Logistic 51
    LogLogistic 51
    LogNormal 52
    NegativeBinomial 52
    Normal 52
    Pareto 52
    PearsonType5 53
    PearsonType6 53
    Poisson 53
    Rectangular 53
    Triangular 54
    Weibull 54
distribution, definition 73
DivRem macro 43
do statements 21

E

ending effect, definition 73
estimated task accuracy, definition 74
estimated task time, definition 74
event, definition 74
Execution menu
    Check for Errors command 87
Exp macro 43
Exponential distribution 59
Exponential macro 49
expressions
    adjustment operators 15
    arithmetic operators 14
    assignment operator 15
    commas 12
    comments 12
    constants 9

do statements 21
for loops 20
foreach statements 23
if-then else statements 18
logical operators 16
macros 11
return statements 13
switch statements 21
variables 10
while statements 19
ExtremeValueTypeA distribution 60
ExtremeValueTypeA macro 50
ExtremeValueTypeB distribution 60
ExtremeValueTypeB macro 50

F

Floor macro 43
for loops 20
force unit, definition 75
foreach statements 23
function criterion, definition 75
function time requirement, definition 75
function, definition 75

G

Gamma distribution 61
Gamma macro 50
Geometric distribution 61
Geometric macro 51
global variables 11
Glossary 71

H

Halt macro 30

I

IEERemainder macro 44
if-then else statements 18
Inverse Gaussian distribution 62
InverseGaussian macro 51

J

job, definition 75

K

Kill macro 30

L

length of run, definition 76
local variables 11
Log macro 44
Log10 macro 44
logical operator 16
Logistic distribution 62
Logistic macro 51
LogLogistic distribution 62
LogLogistic macro 51
Lognormal distribution 63
LogNormal macro 52

M

macros
Abort 27
Abort(array) 28
Abs 41
Acos 41
Asin 41
Atan 41
Atan2 42
Beep 28
Bernoulli 48
Beta 48
BigMul 42
Binomial 49
Ceiling 42
Cos 42
Cosh 43
CreateEntity 28
Direction 29
DiscreteUniform 49
Distance 29
DivRem 43
Exp 43
Exponential 49
ExtremeValueTypeA 50
ExtremeValueTypeB 50
Floor 43
Gamma 50
Geometric 51
Halt 30
IEERemainder 44
in expressions 11
InverseGaussian 51
Kill 30
Log 44
Log10 44
Logistic 51
LogLogistic 51

LogNormal 52
Max 44
Message 30
Min 45
NegativeBinomial 52
Normal 52
Pareto 52
Pause 32
PearsonType5 53
PearsonType6 53
Poisson 53
Pow 45
PrintOutput 32
Random 33
RandomInteger 33
RandomSeed 33
Rectangular 53
Resume 34
Resume(array) 34
Round 45
RunNumber 34
SendExternalVariable 35
Sign 45
Sin 46
Sinh 46
Sqrt 46
Start 35
Stop 35
Stop(array) 36
Suspend 36
Suspend(array) 39
Tan 46
Tanh 47
ToString 39
Triangular 54
TriggerSnapshot 39
WallClock 40
Weibull 54
math macros
Abs 41
Acos 41
Asin 41
Atan 41
Atan2 42
BigMul 42
Ceiling 42
Cos 42
Cosh 43
DivRem 43
Exp 43
Floor 43
IEERemainder 44
Log 44
Log10 44
Max 44
Min 45
Pow 45
Round 45
Sign 45
Sin 46
Sinh 46
Sqrt 46
Tan 46
Tanh 47
Max macro 44
max number of systems, definition 76
mean accuracy, definition 76
mean time expression, definition 76
Message macro 30
Min macro 45
minimum number of systems, definition 76
mission accuracy criterion, definition 76
mission criterion, definition 77
mission description, definition 77
mission name 77
mission time criterion 77
mission time requirement, definition 77
MMH/System, definition 77
model macros
Abort 27
Abort(array) 28
Beep 28
CreateEntity 28
Direction 29
Distance 29
Halt 30
Kill 30
Message 30
Pause 32
PrintOutput 32
Random 33
RandomInteger 33
RandomSeed 33
Resume 34
Resume(array) 34
RunNumber 34
SendExternalVariable 35
Start 35
Stop 35
Stop(array) 36
Suspend 36
Suspend(array) 39
ToString 39
TriggerSnapshot 39
WallClock 40

N

NegativeBinomial distribution 63
NegativeBinomial macro 52
new systems, definition 77
Normal distribution 64
Normal macro 52
number of times to run the mission, definition . 78
number per departure group, definition 78

O

OP Tempo, definition 78
operators
    adjustment 15
    arithmetic 14
    assignment 15
    logical 16
overall workload, definition 78

P

Pareto distribution 64
Pareto macro 52
path, definition 78
Pause macro 32
PearsonType5 distribution 65
PearsonType6 distribution 65
PearsonType6 macro 53
PerasonType5 macro 53
Planned Activity, definition 79
Poisson distribution 66
Poisson macro 53
Pow macro 45
primary operator, definition 79
PrintOutput macro 32
probability distributions
    Bernoulli 57
    Beta 58
    Binomial 58
    continuous 55
    discrete 55
    DiscreteUniform 59
    Exponential 59
    ExtremeValueTypeA 60
    ExtremeValueTypeB 60
    Gamma 61
    Geometric 61
    InverseGaussian 62
    Logistic 62
    LogLogistic 62
    Lognormal 63
    NegativeBinomial 63
    Normal 64
    overview 55
    Pareto 64
    PearsonType5 65
    PearsonType6 65
    Poisson 66
    Rectangular 66
    references 57
    scale parameter 56
    shape parameter 56
    Triangular 67
    Weibull 67
probability of success, definition 79

R

Random macro 33
random number seed, definition 79
RandomInteger macro 33
RandomSeed macro 33
rank, definition 79
Rectangular distribution 66
Rectangular macro 53
release condition, definition 79
repeating segments, definition 79
replacement systems, definition 80
resource-interface channels, definition 80
Resume macro 34
Resume(array) macro 34
return statements 13
role, definition 80
Round macro 45
RunNumber macro 34

S

scenario description, definition 80
scenario name, definition 80
scoping, variables 11
segment cancellation time, definition 81
segment duration time, definition 81
segment priority, definition 81
segment repeat mean time, definition 81
segment start day, definition 81
segment start time, definition 82
SendExternalVariable macro 35
shift data, definition 82
shift length, definition 82
shifts per day, definition 82
Sign macro 45
Sin macro 46
Sinh macro 46

specialty, definition 82
Sqrt macro 46
Start macro 35
Stop macro 35
Stop(array) macro 36
subsystem equipment name, definition 82
subsystem, definition 83
Suspend macro 36
Suspend(array) macro 39
switch statements 21

T

Tan macro 46
Tanh macro 47
task accuracy probability, definition 83
task criterion, definition 83
task mean accuracy, definition 83
task name, definition 83
task priority, definition 84
task standard deviation, definition 84
task time requirement, definition 84
task, definition 83
time between departures, definition 84
time format, definition 84
time requirement, definition 85
timing and accuracy tab
    mean accuracy 76
ToString macro 39
Triangular distribution 67
Triangular macro 54
TriggerSnapshot macro 39

U

uniform distribution 66
unplanned activity, definition 85

V

variable, definition 85
variables
    defined 10
    global 11
    local 11
    scoping 11

W

WallClock macro 40
warfighter, definition 85
Weibull distribution 67
Weibull macro 54
while statements 19