MapleMBSE Application Guide

Contents

Introduction

MapleMBSE Application Guide Overview

MapleMBSE™ gives an intuitive, spread-sheet based user interface for entering detailed system design definitions, which include structures, behaviors, requirements, and parametric constraints.

The Application directory of your MapleMBSE installation contains six applications. Each of the chapters in this guide corresponds to one of the applications:

Chapter	Application Name	Description
1	Working With Blocks in MapleMBSE	The first application uses the TWCSysML-Structure.mse file to demonstrate the use of blocks in MapleMBSE
2	Creating a Model in MapleMBSE (Fitness Tracker Model)	This model uses the **TWCSysML-Model.mse** and **TWCSysML-ModelActivity.mse** files to demonstrate how to create a model in MapleMBSE which can be exported to the Teamwork Cloud
3	Working With State Machine Diagrams in MapleMBSE	The example in this chapter defines how to create states, define their transitions and the events that trigger these transitions using MapleMBSE.
4	Count Down Timer Model	This chapter contains a model of Countdown Timer that uses **TWCSysML-Timer.mse** to create a simulatable Timer model.
5	Turbofan Engine Model	This example model is used to identify design points of a turbofan engine. MapleMBSE and Cameo Systems Modeler™ were used to create a turbofan example model
6	UAV Model	This model uses Object Oriented System Engineering Methodology (OOSEM) to design a conceptual model of an Unmanned Aerial Vehicle (UAV).
7	FMEA Template	This model is used to perform FMEA analysis by accessing SysML model elements from the Teamwork Cloud server.

Related Products

MapleMBSE 2019 requires the following products:

- Microsoft® Excel® 2010 Service Pack 2, Excel 2016 or Excel 2019.

- Oracle® Java® SE Runtime Environment 8.

Note: MapleMBSE looks for a Java Runtime Environment in the following order:

1) If you use the -vm option specified in **OSGiBridge.init** (not specified by default)

2) If your environment has a system JRE (meaning either: JREs specifed by the environment variables JRE_HOME and JAVA_HOME in this order, or a JRE specified by the Windows Registry (created by JRE installer)), MapleMBSE will use it.

3) The JRE installed in the MapleMBSE installation directory.

If you are using IBM® Rational® Rhapsody® with MapleMBSE, the following version is supported: Rational Rhapsody Version 8.15

- Teamwork Cloud™ server 18.5 SP3 or 19.0 SP2
- Note that the architecture of the supported non-server products (that is, 32-bit or 64-bit) must match the architecture of your MapleMBSE architecture.

Related Resources

Resource	Description
MapleMBSE Installation Guide	System requirements and installation instructions for MapleMBSE. The **MapleMBSE Installation Guide** is available in the **Install.html** file located either on your MapleMBSE installation DVD or the folder where you installed MapleMBSE.
MapleMBSE User Guide	Instructions for using MapleMBSE software. The **MapleMBSE User Guide** is available in the folder where you installed MapleMBSE.
MapleMBSE Configuration Guide	

For additional resources, visit **http://www.maplesoft.com/site_resources**.

Getting Help

To request customer support or technical support, visit **http://www.maplesoft.com/support**.

Customer Feedback

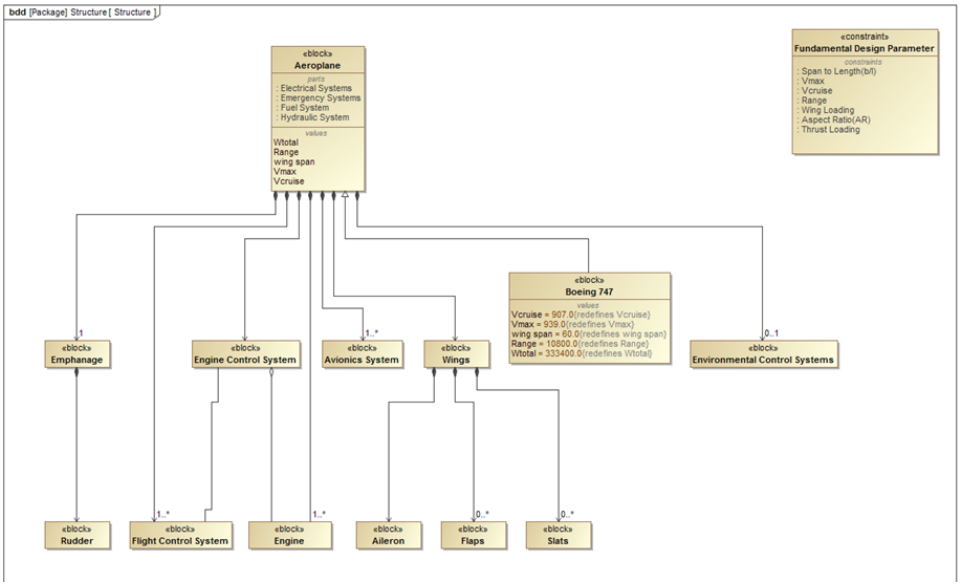
Maplesoft welcomes your feedback. For comments related to the MapleMBSE product documentation, contact doc@maplesoft.com.

Copyrights

1 Blocks in MapleMBSE

1.1 Blocks Table

The block diagram shown below is created using MapleMBSE and syncing it to the Teamwork Cloud. This chapter will explain how to work with blocks in MapleMBSE.



This example is created with the following package structure:

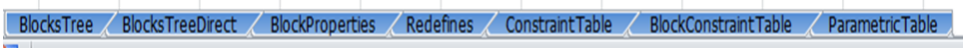
Model

+ Structure

The list of features available in MapleMBSE to define blocks are:

- Association
- Aggregation
- Composition
- Generalization
- OwnedEnd Multiplicty
- Constraint
- Property

- Value
- Operations
- Redefine Value

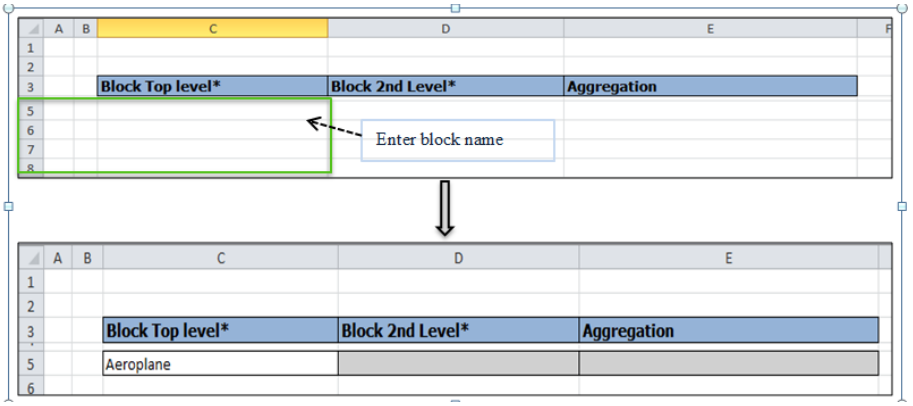


The configuration file, **TWCSysML-Structure.mse** defines *seven* worksheet templates to work with blocks:

- The **BlocksTree** and **BlocksTreeDirect** worksheets are used to create blocks and their relationships.
- The **BlockProperties** worksheet is used to create generalizations, values and operations.
- The **Redefines** worksheet is used to specify values and redefine values to blocks.
- The **ConstraintTable** worksheet is used to create parameters, opaque expressions and define constraint blocks.
- **BlockConstraintTable** is used to create a direct association between Blocks and Constraint Blocks.
- **Parametric Table** is used to create a binding connector between the constraint parameters.

Creating a Block

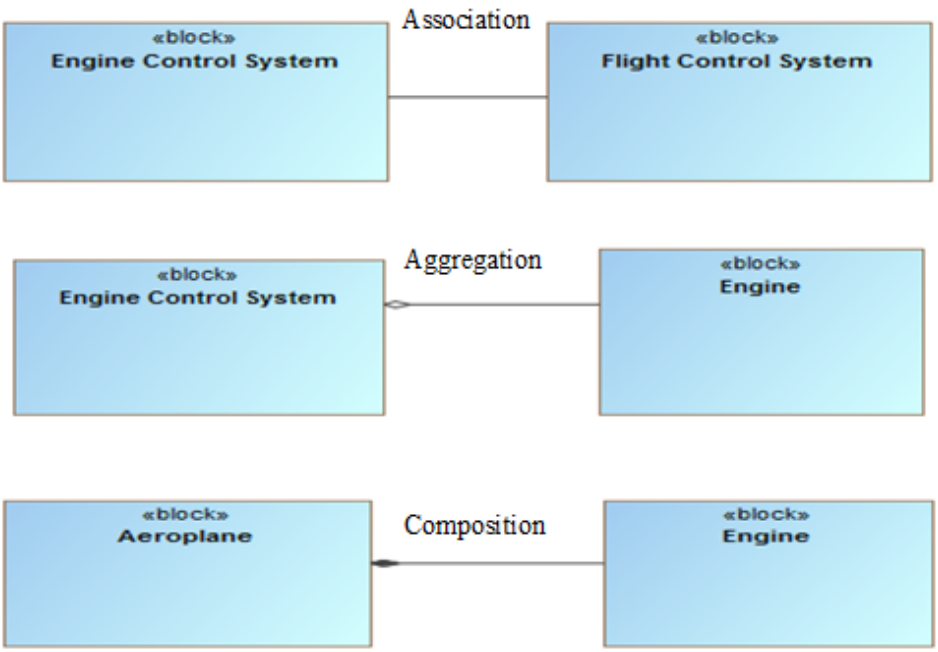
To create a block, enter a name for the block in the column C insertion area (the **Block Top Level** column) as shown below. A block called **Aeroplane** is created.



To create a relation between blocks, they must first be created in the **Block Top Level** column before they can be added in the second level.

Blocks can be created in all worksheets except for the C**onstraintTable** worksheet.

1.2 Creating Association, Aggregation and Composition



To create relations without direction, use the **BlocksTree** worksheet. The blocks need to be created as shown below.

To create Association relations:

1. Enter the block name in the **Block Top Level** column.

Block Top level*	Block 2nd Level*	Aggregation
Aeroplane		
Engine		
Engine Control System		
Flight Control System		

2. The row is highlighted as a duplicate key to indicate the block already exists. Enter the related block name in the **Block 2nd Level** column, in the same row.

Block Top level*	Block 2nd Level*	Aggregation
Aeroplane		
Engine		
Engine Control System	Duplicated Key	
Flight Control System		
Engine Control System		

3. MapleMBSE checks if the entry is valid by comparing it with existing blocks and will add **none** in the **Aggregation** column by default.

Block Top level*	Block 2nd Level*	Aggregation
Aeroplane		
Engine		
Engine Control System		
Flight Control System		
Engine Control System	Flight Control System	none

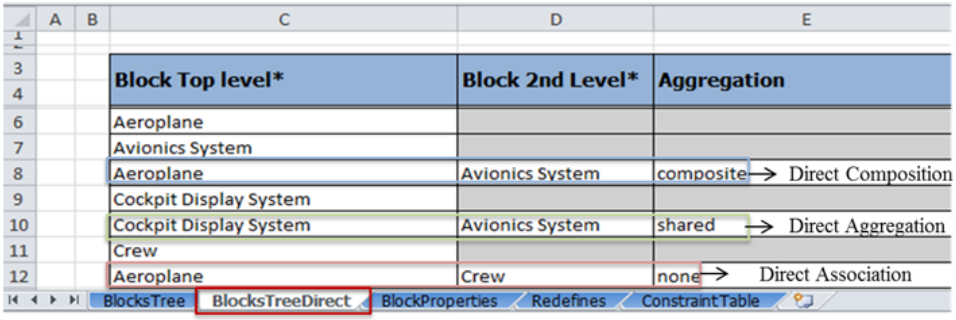
To create Aggregation and Composition relations, follow the previous steps by entering the owned end block (the class that has an association owned by another class) in column C, replace **none** with **composite** in the **Aggregation** column to create a composition relation and **shared** to create an aggregation relation.

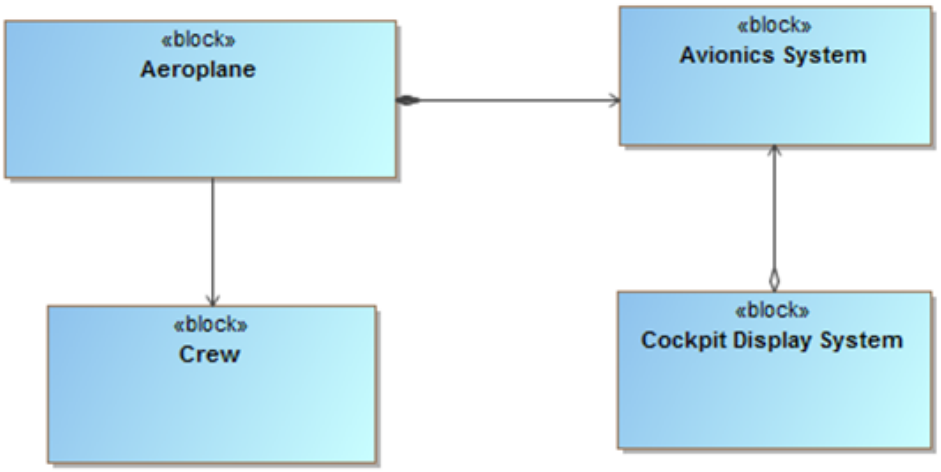
B	C	D	E	
	Block Top level*	**Block 2nd Level***	**Aggregation**	
	Aeroplane			
	Engine			
	Engine Control System			
	Flight Control System			
	Aeroplane	Engine	composite	→ Composition
	Engine Control System	Engine	shared	→ Aggregation
	Flight Control System	Engine Control System	none	→ Association

BlocksTree | BlocksTreeDirect | BlockProperties | Redefines | ConstraintTable

1.3 Creating Direct Association, Aggregation and Composition

Use the **BlocksTreeDirect** worksheet to create relations with direction. Both tables are similar in defining relations, the type of relation differs based on the entry in the **Aggregation** column. Enter the class name in the **Block Top Level** column and enter the name of the **Attribute** class in the **Block 2nd Level** column and specify the aggregation type. The figure below shows relations between blocks with navigability.





The following table shows the necessary information needed to create a relation between blocks and their corresponding worksheet. The **Class** and **Attribute Class** columns imply that the class and its related class should be created first and then the respective aggregation type.

Worksheet	Type	Class	Attribute Class	Aggregation
BlocksTree	Association	x	x	None
	Aggregation	x	x	shared
	Composition	x	x	composite
BlocksTreeDirect	Direct Association	x	x	None
	Direct Aggregation	x	x	shared
	Direct Composition	x	x	composite

To represent multiplicity, at the Association level, enter a value for the respective blocks in the **Multiplicity** column as shown below.

Multiplicity
0
1
0..1
0..*
1..*

1.4 Block Generalization, Values and Operation

To generalize a block, enter the name of the generalizing block in the **Block Top Level** column of the **BlockProperties** worksheet and a corresponding value in the **Generalization Block** column.

	Block Top Level	Generalization Block	Value	Operation
5	Aeroplane			
6	Boeing 747			
7	Boeing 747	Aeroplane		

Use the same worksheet to add a value property to a block. Enter the block name in the **Block Top Level** column and then enter the value in the **Value** column.

	Block Top Level	Generalization Block	Value	Operation
5	Aeroplane			
6	Aeroplane		Wtotal	
7	Aeroplane		wing span	
8	Aeroplane		Vmax	
9	Aeroplane		Vcruise	
10	Aeroplane		Range	

Similarly, to add operations to the blocks, enter the block name in the **Block Top Level** column and the operation name in the **Operation** column.

Block Top Level	Generalization Block	Value	Operation
Engine Control System			
Engine Control System			monitor engine temperature
Engine Control System			monitor engine pressure
Engine Control System			control fuel flow

In the **Redefines** worksheet, to enter a numerical value for Value Property use the **Value** column, as shown below.

Block	Value Property	Value	Redefine	
			Block	Property
Aeroplane				
Aeroplane	Range			
Aeroplane	Vcruise			
Aeroplane	Vmax			
Aeroplane	wing span			
Aeroplane	Wtotal			
Boeing 747				
Boeing 747	Range	10800	Aeroplane	Range
Boeing 747	Vcruise	907	Aeroplane	Vcruise
Boeing 747	Vmax	939	Aeroplane	Vmax
Boeing 747	wing span	60	Aeroplane	wing span
Boeing 747	Wtotal	333400	Aeroplane	Wtotal

To redefine a property of an existing block, type a new value in the **Value** column along with information about the block from which the value is redefined. For example, **Aeroplane** has value properties: **Range**, **Vcruise**, **Vmax**, **wing span** and **Wtotal.** These properties are not defined with numerical values, as shown above (these fields can hold numerical values). The **Boeing 747** block is generalized to **Aeroplane**. To redefine the values from **Aeroplane** to **Boeing 747**, enter the same value for **Boeing 747** properties as that of **Aeroplane**. In the **Value** column, enter the desired values. Now to redefine, enter the block from which the value is redefined and the name of the value being redefined as shown below.

	A	B	C	D	E	F	G
1							
2			Block	Value Property	Value	Redefine	
3						Block	Property
5			Aeroplane				
6			Aeroplane	Range	Values to be		
7			Aeroplane	Vcruise	redefined from		
8			Aeroplane	Vmax	Aeroplane	Redefined Value	and
9			Aeroplane	wing span	Values redefined to	Block name	
10			Aeroplane	Wtotal	Boeing 747		
11			Boeing 747				
12			Boeing 747	Range	10800	Aeroplane	Range
13			Boeing 747	Vcruise	907	Aeroplane	Vcruise
14			Boeing 747	Vmax	939	Aeroplane	Vmax
15			Boeing 747	wing span	60	Aeroplane	wing span
16			Boeing 747	Wtotal	333400	Aeroplane	Wtotal

1.5 Constraint Blocks

The process for creating constraint blocks, relations and parameters is similar to that of working with blocks in the previous section.

Constraint Block Top Level	Constraint Block 2nd Level*	Constraint Parameters	Constraint Name	Constraint Block	Specification Name	OpaqueExpression
Aspect Ratio						
Aspect Ratio		AR				
Aspect Ratio			ratio	Aspect Ratio		
Aspect Ratio			ratio	Aspect Ratio	eq	b^2/s
Fundamental Design Parameter						
Fundamental Design Parameter	Aspect Ratio					

In the **Constraint Block Top Level** column, enter a constraint block and its breakdown in the **Constraint Block 2nd Level** column. This creates a direct composition relation between the blocks. In order to create different relations between the constraint blocks the configuration file has to be edited. To create parameters, enter the respective block in the **Constraint Block Top Level** column and the parameter name in the **Constraint Parameters** column. To add an equation to a constraint block, enter the block name followed by the name of the constraint in the **Constraint Name** column, as shown above. Enter the constraint block name in the **Constraint Block Top Level** column and a name for the specification equation in the **Specification** column. MapleMBSE accepts the entry. The corresponding field in the **Opaque Expression** column is empty. Enter an expression, as shown in the figure.

To create a direct association between the blocks and *Constraint Blocks* select the BlockConstraintTable worksheet. Next, enter the block name in the **Block Name** column and *Constraint Block* in the **Constraint Block Name** column, as shown below.

Block Name	Constraint Block Name
Analysis Context	
Fundamental Design Parameter	
Analysis Context	Fundamental Design Parameter

To create a binding connector between the parameters of the Constraint Blocks, you must first open the ParametricTable worksheet. Enter the *Constraint Block* and the parameter of the constraint that has to be connected in the **Constraint Parameter Column**, followed by the *Constraint Block* name and the target parameter in the respective column. MapleMBSE will automatically create a binding connector between the two parameters of the constraint blocks specified.

		Binding Connector	
Constraint Block	Constraint Parameter	Constraint Block	Constraint Parameter
Aspect Ratio			
Aspect Ratio	AR		
Fundamental Design Parameter			
Fundamental Design Parameter	AR		
Aspect Ratio	AR	Fundamental Design Parameter	AR
Fundamental Design Parameter	AR	Aspect Ratio	AR

2 The Fitness Tracker Model

The Excel Workbook template, **TWCSysML-Model.xlsx**, arranges the display of the elements in worksheets as defined in the configuration files.

The Package structure of the model is displayed in the **Packages** worksheet.

The Requirements packages are defined hierarchically; defining a top-level requirement, decomposing the requirements into groups and finally stating the requirements.

Once the requirements are defined, actors and their interactions with the system are created in the **Actors** and **UseCases** worksheets.

The **BlockTree** and **BlockProperties** worksheets are used to display information about the system context, specifications and relations.

The **BlockConnectorTable** and **BlockPropertyTable** worksheets create connections between block properties.

Once the structural aspects are defined, the system's behavior are defined by using the **TWCSysML-ModelActivity.mse** configuration file.

This example was created with the following package structure:

Model

- Requirements

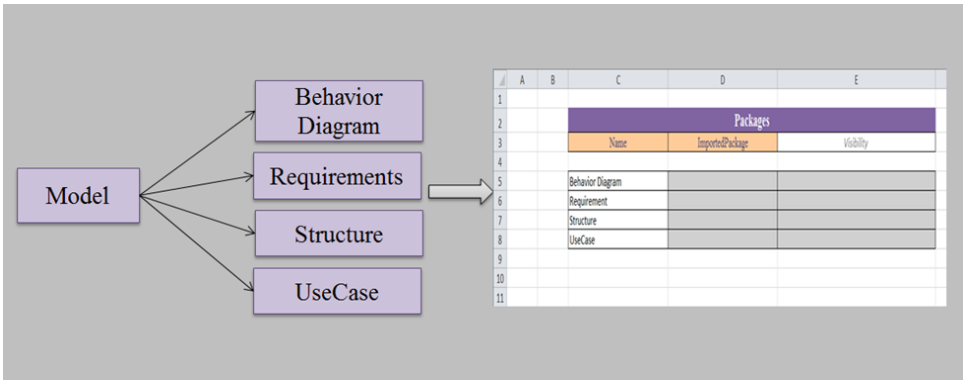
- Use Case

- Structure

- Behavior

2.1 Packages

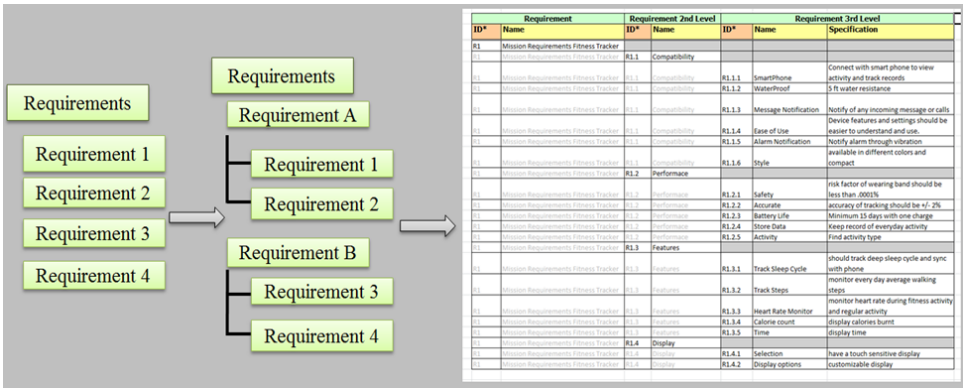
The **Packages** worksheet is used to organize the model elements into respective *Packages*. The user can create packages by specifying a name for the package under the **Name** column in the **Packages** worksheet. *Packages* are created as shown in the figure below. The configuration (.mse) file is configured in such as way so that when a user begins working directly in a worksheet, without creating any packages beforehand, the packages are automatically created and elements are displayed under the packages corresponding to the worksheet.



2.2 Requirements Table

The requirements defined for a system are used to identify the behavior, constraints, system specifications, etc. for which the system is modeled. Requirements can be categorized or grouped based on their definition of the system such as: performance, functional, constraints, etc.

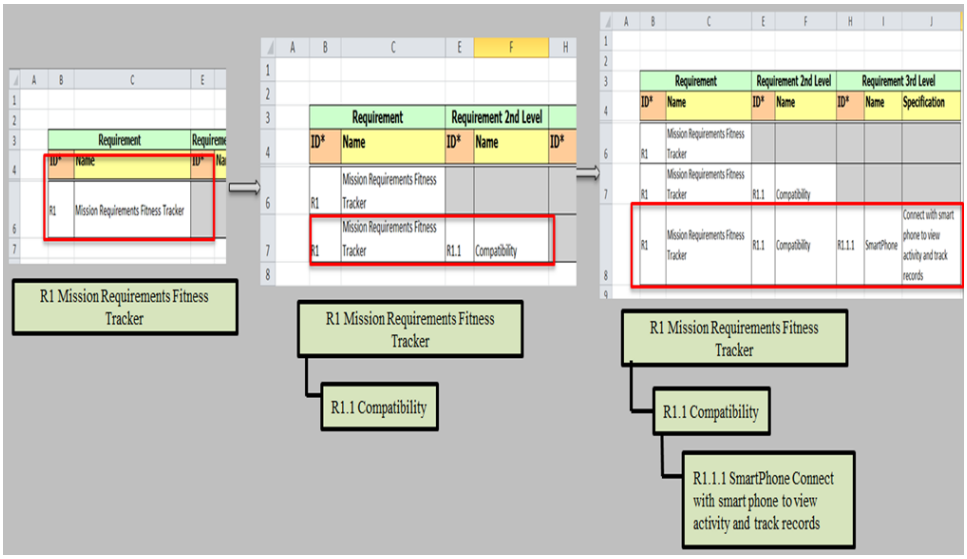
This example was created with requirements in three levels, as shown in the Excel file below. The number of levels and appearance of the **Requirements** worksheet is controlled by the configuration (.mse) file and can be changed by editing the configuration file.



Requirement		Requirement 2nd Level		Requirement 3rd Level		
ID*	Name	ID*	Name	ID*	Name	Specification
R1	Mission Requirements Fitness Tracker					
R1	Mission Requirements Fitness Tracker	R1.1	Compatibility			
R1	Mission Requirements Fitness Tracker	R1.1	Compatibility	R1.1.1	SmartPhone	Connect with smart phone to view activity and track records
R1	Mission Requirements Fitness Tracker	R1.1	Compatibility	R1.1.2	WaterProof	5 ft water resistance
R1	Mission Requirements Fitness Tracker	R1.1	Compatibility	R1.1.3	Message Notification	Notify of any incoming message or calls
R1	Mission Requirements Fitness Tracker	R1.1	Compatibility	R1.1.4	Ease of Use	Device features and settings should be easier to understand and use.
R1	Mission Requirements Fitness Tracker	R1.1	Compatibility	R1.1.5	Alarm Notification	Notify alarm through vibration
R1	Mission Requirements Fitness Tracker	R1.1	Compatibility	R1.1.6	Style	available in different colors and compact
R1	Mission Requirements Fitness Tracker	R1.2	Performace			
R1	Mission Requirements Fitness Tracker	R1.2	Performace	R1.2.1	Safety	risk factor of wearing band should be less than .0001%
R1	Mission Requirements Fitness Tracker	R1.2	Performace	R1.2.2	Accurate	accuracy of tracking should be +/- 2%
R1	Mission Requirements Fitness Tracker	R1.2	Performace	R1.2.3	Battery Life	Minimum 15 days with one charge
R1	Mission Requirements Fitness Tracker	R1.2	Performace	R1.2.4	Store Data	Keep record of everyday activity
R1	Mission Requirements Fitness Tracker	R1.2	Performace	R1.2.5	Activity	Find activity type
R1	Mission Requirements Fitness Tracker	R1.3	Features			
R1	Mission Requirements Fitness Tracker	R1.3	Features	R1.3.1	Track Sleep Cycle	should track deep sleep cycle and sync with phone
R1	Mission Requirements Fitness Tracker	R1.3	Features	R1.3.2	Track Steps	monitor every day average walking steps
R1	Mission Requirements Fitness Tracker	R1.3	Features	R1.3.3	Heart Rate Monitor	monitor heart rate during fitness activity and regular activity
R1	Mission Requirements Fitness Tracker	R1.3	Features	R1.3.4	Calorie count	display calories burnt
R1	Mission Requirements Fitness Tracker	R1.3	Features	R1.3.5	Time	display time
R1	Mission Requirements Fitness Tracker	R1.4	Display			
R1	Mission Requirements Fitness Tracker	R1.4	Display	R1.4.1	Selection	have a touch sensitive display
R1	Mission Requirements Fitness Tracker	R1.4	Display	R1.4.2	Display options	customizable display

Creating Requirements

Requirements contain a unique **ID**, **Name** and **Specification** field to identify and name each requirement with a brief description.



To enter a new requirement:

1. Enter an ID for a top level requirement in the **ID** column, as shown above. MapleMBSE checks for duplicate entries and adds a row for the corresponding ID, enabling the user to enter a name and specification for the requirement.
2. To create a second level requirement, use the same ID and name as for the top-level requirement. MapleMBSE will detect it as a duplicate entry and highlight it as a duplicate key. Type an ID for the requirement in the **ID** column, of the **Requirement 2nd Level** section (column E), as shown above. MapleMBSE considers this to be a unique entry and enables the corresponding row to accept a name and description for the requirement.
3. To create a third-level Requirement, follow step 2, then enter a new ID in column H.

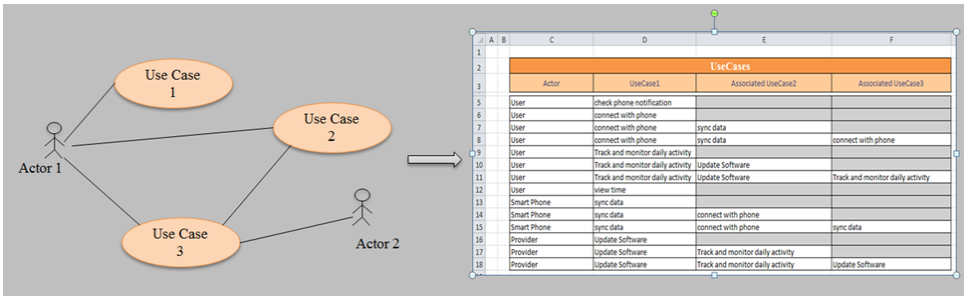
Follow the above steps to create any number of requirements. Excel identifies the **ID** columns as text format fields. The figure below shows the requirements created for the Fitness Tracker model, using the steps above.

	Requirement		Requirement 2nd Level		Requirement 2nd Level		
	ID*	Name	ID*	Name	ID*	Name	Specification
	R1	Mission Requirements Fitness Tracker					
	R1	Mission Requirements Fitness Tracker	R1.1	Compatibility			
	R1	Mission Requirements Fitness Tracker	R1.1	Compatibility	R1.1.1	SmartPhone	Connect with smart phone to view activity and track records
	R1	Mission Requirements Fitness Tracker	R1.1	Compatibility	R1.1.2	WaterProof	5 ft water resistance
	R1	Mission Requirements Fitness Tracker	R1.1	Compatibility	R1.1.3	Message Notification	Notify of any incoming message or calls
	R1	Mission Requirements Fitness Tracker	R1.1	Compatibility	R1.1.4	Ease of Use	Device features and settings should be easier to understand and use.
	R1	Mission Requirements Fitness Tracker	R1.1	Compatibility	R1.1.5	Alarm Notification	Notify alarm through vibration
	R1	Mission Requirements Fitness Tracker	R1.1	Compatibility	R1.1.6	Style	available in different colors and compact
	R1	Mission Requirements Fitness Tracker	R1.2	Performace			
	R1	Mission Requirements Fitness Tracker	R1.2	Performace	R1.2.1	Safety	risk factor of wearing band should be less than .0001%
	R1	Mission Requirements Fitness Tracker	R1.2	Performace	R1.2.2	Accurate	accuracy of tracking should be +/- 2%
	R1	Mission Requirements Fitness Tracker	R1.2	Performace	R1.2.3	Battery Life	Minimum 15 days with one charge
	R1	Mission Requirements Fitness Tracker	R1.2	Performace	R1.2.4	Store Data	Keep record of everyday activity
	R1	Mission Requirements Fitness Tracker	R1.2	Performace	R1.2.5	Activity	Find activity type
	R1	Mission Requirements Fitness Tracker	R1.3	Features			
	R1	Mission Requirements Fitness Tracker	R1.3	Features	R1.3.1	Track Sleep Cycle	should track deep sleep cycle and sync with phone
	R1	Mission Requirements Fitness Tracker	R1.3	Features	R1.3.2	Track Steps	monitor every day average walking steps
	R1	Mission Requirements Fitness Tracker	R1.3	Features	R1.3.3	Heart Rate Monitor	monitor heart rate during fitness activity and regular activity
	R1	Mission Requirements Fitness Tracker	R1.3	Features	R1.3.4	Calorie count	display calories burnt
	R1	Mission Requirements Fitness Tracker	R1.3	Features	R1.3.5	Time	display time
	R1	Mission Requirements Fitness Tracker	R1.4	Display			
	R1	Mission Requirements Fitness Tracker	R1.4	Display	R1.4.1	Selection	have a touch sensitive display
	R1	Mission Requirements Fitness Tracker	R1.4	Display	R1.4.2	Display options	customizable display

2.3 Use Case Table

The **Use Case** table describes the goals and interactions of the system model with external users (stakeholders).

To create a use case table, the actors of the system are identified, then the goals of the system and other functionality expected by the user.



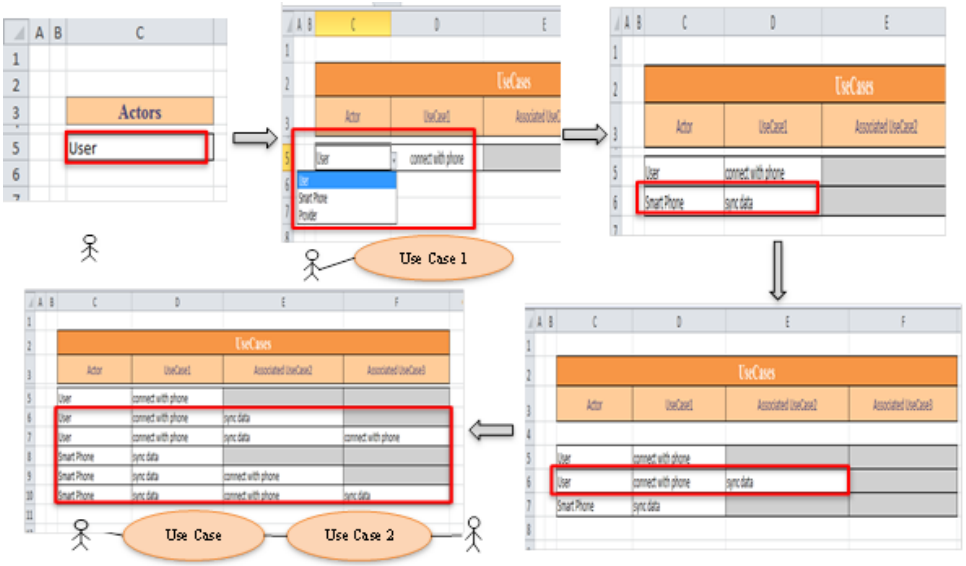
UseCases			
Actor	UseCase1	Associated UseCase2	Associated UseCase3
User	check phone notification		
User	connect with phone		
User	connect with phone	sync data	
User	connect with phone	sync data	connect with phone
User	Track and monitor daily activity		
User	Track and monitor daily activity	Update Software	
User	Track and monitor daily activity	Update Software	Track and monitor daily activity
User	view time		
Smart Phone	sync data		
Smart Phone	sync data	connect with phone	
Smart Phone	sync data	connect with phone	sync data
Provider	Update Software		
Provider	Update Software	Track and monitor daily activity	
Provider	Update Software	Track and monitor daily activity	Update Software

Creating a Use Case Table

Use cases and actors are identified by unique names. The configuration file is created in such a way that two different tables are needed to create the use case table. The **Actors** worksheet is used to list the identified actors of the system. The **UseCases** worksheet is then used to create the interaction between actors and use cases.

To create a Use Case table:

1. Create actors in the **Actors** worksheet as shown below.



2. In the **UseCases** worksheet, type the name of the actor to create a use case or select a name from the list. Type the use case in the **UseCase1** column as shown above.
3. To relate use cases, enter the actor name and corresponding use case in columns C and D respectively. MapleMBSE will highlight this as a duplicate key. Enter the other use case in the **Associated UseCase2** column (column E). This entry is considered valid and rows are automatically created to show that the association is bidirectional.

The **Use Case** table created for the Fitness Tracker is shown below. The **Associated UseCase3** column is automatically generated by MapleMBSE based on the input in the other columns. To associate use cases, they must already exist in the **UseCase1** column.

	A	B	C	D	E	F
1						
2			UseCases			
3			Actor	UseCase1	Associated UseCase2	Associated UseCase3
5			User	check phone notification		
6			User	connect with phone		
7			User	connect with phone	sync data	
8			User	connect with phone	sync data	connect with phone
9			User	Track and monitor daily activity		
10			User	Track and monitor daily activity	Update Software	
11			User	Track and monitor daily activity	Update Software	Track and monitor daily activity
12			User	view time		
13			Smart Phone	sync data		
14			Smart Phone	sync data	connect with phone	
15			Smart Phone	sync data	connect with phone	sync data
16			Provider	Update Software		
17			Provider	Update Software	Track and monitor daily activity	
18			Provider	Update Software	Track and monitor daily activity	Update Software

2.4 Blocks Table

Blocks are created in a predefined package named, **Structure**. From the configuration file, three worksheets are created:

- **BlockTree** to create blocks and parts,
- **BlockProperties** to create operations, generalizations and to create values for the blocks, and
- **BlockSatisfiesMatrix** to validate the model against the requirements to identify if all requirements have been met.

To make the example model simpler, only direct composition and generalization relations between blocks are used.

Blocks Tree

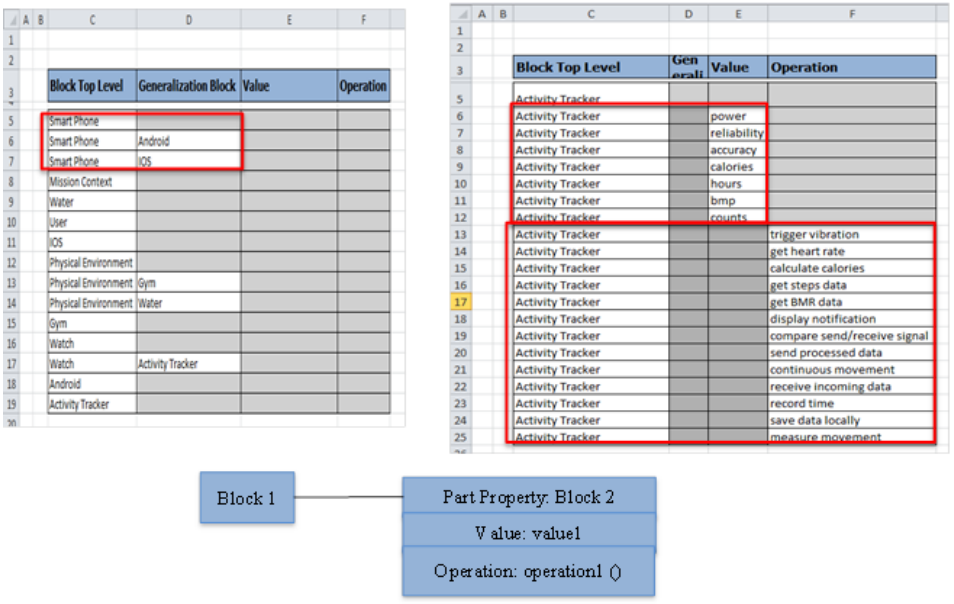
Blocks are identified uniquely by their names and can be accessed between worksheets. To identify the scope and working environment of the system, the mission context table is created using the **BlockTree** and **BlockProperties** worksheets.

Once the system scope is defined, a blackbox specification for the system of interest is created in terms of values and operations. These operations defined for the system are used to work with the behavior of the system defined in a different configuration file.

On defining activities of the system using the behavior configuration file, logical blocks are defined in the same table using **BlocksTree**. Finally, parts of the system are defined at

a physical level to meet the requirements specifications and also to satisfy the behavioral aspect of the system modeled.

1. To create a block, enter a name for the block in the **Block Top Level** column (column C), as shown below. Every unique entry in this column creates a block. Text entered is case sensitive so to create properties for a block in the second level, the block name should be accessed with the same case.



2. To create a direct compostion between blocks or to assign a block as part of another block type, enter the name of the block for which a part has to be created in the **Block Top Level** column followed by the part name in **Block 2nd Level**, as shown above. Now a direct association is created between **Mission Context** and **Activity Tracker**.

3. Blocks can be created at a third level in two ways: similar to adding blocks at the second level, specify the top level block, then the second level block, and finally the third level

block name. The figure below illustrates this way of adding a third level block. Since **Screen** is already a part property of **Activity Tracker**, physically adding a part to **Screen**, as shown in row 9, will automatically create row 6 and vice versa.

	Block Top level*	Block 2nd Level*	Block 3rd Level*
5	Activity Tracker - Physical	Screen	
6	Activity Tracker - Physical	Screen	Capacitive touch Screen
7	Capacitive touch Screen		
8	Screen		
9	Screen	Capacitive touch Screen	

To create generalizations, the **BlockProperties** worksheet is used. Similar to the above step, once blocks are created in the top level column, enter the block name in **Block Top Level** and the generalizing block in the **Generalization Block** column (cell D6). In the table, **Android** and **IOS** are generalized to **Smartphone**.

	Block Top Level	Generalization Block	Value	Operation
5	Smart Phone			
6	Smart Phone	Android		
7	Smart Phone	IOS		
8	Mission Context			
9	Water			
10	User			
11	IOS			
12	Physical Environment			
13	Physical Environment	Gym		
14	Physical Environment	Water		
15	Gym			
16	Watch			
17	Watch	Activity Tracker		
18	Android			
19	Activity Tracker			

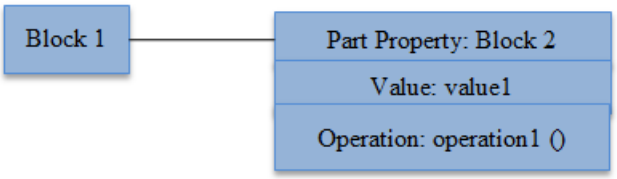
To create the value and operation property of a block, in the **BlockProperties** worksheet enter the name of the block that you want to assign a value. Since the block already exists, the row is highlighted as a duplicate key. Type the value in the **Value** column (column E),

as shown below, to add a value to the block, **Activity Tracker** for this example. Notice cells E6 to E12 have values assigned to **Activity Tracker**.

	A	B	C	D	E	F
1						
2						
3			**Block Top Level**	**Gen erali**	**Value**	**Operation**
5			Activity Tracker			
6			Activity Tracker		power	
7			Activity Tracker		reliability	
8			Activity Tracker		accuracy	
9			Activity Tracker		calories	
10			Activity Tracker		hours	
11			Activity Tracker		bmp	
12			Activity Tracker		counts	
13			Activity Tracker			trigger vibration
14			Activity Tracker			get heart rate
15			Activity Tracker			calculate calories
16			Activity Tracker			get steps data
17			Activity Tracker			get BMR data
18			Activity Tracker			display notification
19			Activity Tracker			compare send/receive signal
20			Activity Tracker			send processed data
21			Activity Tracker			continuous movement
22			Activity Tracker			receive incoming data
23			Activity Tracker			record time
24			Activity Tracker			save data locally
25			Activity Tracker			measure movement

In a single row for the block, either value or operation can be assigned to it. To assign operation to a block, a similar procedure is followed. Enter the block to which an operation has to be created in the **Block Top Level** column and enter the operation name in the **Operation** column (column F), as shown below.

	Block Top Level	Gen erali	Value	Operation
5	Activity Tracker			
6	Activity Tracker		power	
7	Activity Tracker		reliability	
8	Activity Tracker		accuracy	
9	Activity Tracker		calories	
10	Activity Tracker		hours	
11	Activity Tracker		bmp	
12	Activity Tracker		counts	
13	Activity Tracker			trigger vibration
14	Activity Tracker			get heart rate
15	Activity Tracker			calculate calories
16	Activity Tracker			get steps data
17	Activity Tracker			get BMR data
18	Activity Tracker			display notification
19	Activity Tracker			compare send/receive signal
20	Activity Tracker			send processed data
21	Activity Tracker			continuous movement
22	Activity Tracker			receive incoming data
23	Activity Tracker			record time
24	Activity Tracker			save data locally
25	Activity Tracker			measure movement

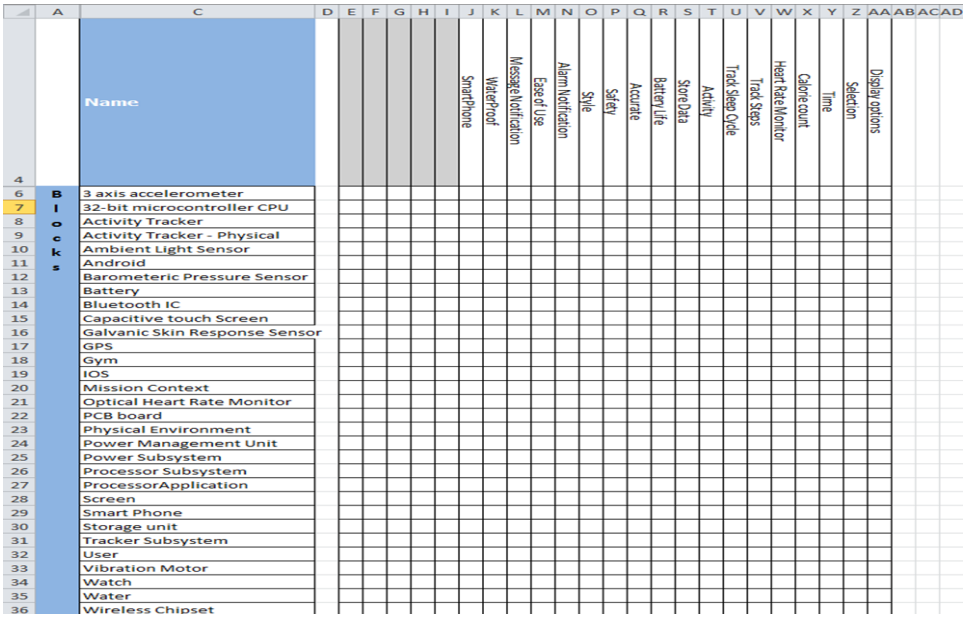


Using the steps mentioned above, the **Activity Tracker** is created and the block table at the physical level is shown while the rest of the inputs are filtered.

Block Top level*	Block 2nd Level*	Block 3rd Level*
3 axis accelerometer		
32-bit microcontroller CPU		
Activity Tracker - Physical		
Activity Tracker - Physical	Power Subsystem	
Activity Tracker - Physical	Power Subsystem	Battery
Activity Tracker - Physical	Power Subsystem	Power Management Unit
Activity Tracker - Physical	Processor Subsystem	
Activity Tracker - Physical	Processor Subsystem	32-bit microcontroller CPU
Activity Tracker - Physical	Processor Subsystem	Bluetooth IC
Activity Tracker - Physical	Processor Subsystem	PCB board
Activity Tracker - Physical	Processor Subsystem	ProcessorApplication
Activity Tracker - Physical	Processor Subsystem	Vibration Motor
Activity Tracker - Physical	Processor Subsystem	Wireless Chipset
Activity Tracker - Physical	Screen	
Activity Tracker - Physical	Screen	Capacitive touch Screen
Activity Tracker - Physical	Tracker Subsystem	
Activity Tracker - Physical	Tracker Subsystem	3 axis accelerometer
Activity Tracker - Physical	Tracker Subsystem	Ambient Light Sensor
Activity Tracker - Physical	Tracker Subsystem	Barometeric Pressure Sensor
Activity Tracker - Physical	Tracker Subsystem	Galvanic Skin Response Sensor
Activity Tracker - Physical	Tracker Subsystem	Optical Heart Rate Monitor
Ambient Light Sensor		
Barometeric Pressure Sensor		
Battery		
Bluetooth IC		
Capacitive touch Screen		
Galvanic Skin Response Sensor		
Optical Heart Rate Monitor		
PCB board		
PCB board	Storage unit	
Power Management Unit		
Power Subsystem		
Power Subsystem	Battery	
Power Subsystem	Power Management Unit	
Processor Subsystem		
Processor Subsystem	32-bit microcontroller CPU	
Processor Subsystem	Bluetooth IC	
Processor Subsystem	PCB board	
Processor Subsystem	PCB board	Storage unit
Processor Subsystem	ProcessorApplication	
Processor Subsystem	ProcessorApplication	32-bit microcontroller CPU
Processor Subsystem	Vibration Motor	
Processor Subsystem	Wireless Chipset	
ProcessorApplication		
ProcessorApplication	32-bit microcontroller CPU	
Screen		
Screen	Capacitive touch Screen	
Storage unit		
Tracker Subsystem		
Tracker Subsystem	3 axis accelerometer	
Tracker Subsystem	Ambient Light Sensor	
Tracker Subsystem	Barometeric Pressure Sensor	
Tracker Subsystem	Galvanic Skin Response Sensor	
Tracker Subsystem	Optical Heart Rate Monitor	
Vibration Motor		
Wireless Chipset		

Block Satisfaction Matrix

The **Block Satisfaction Matrix** is used to verify whether the blocks created satisfy the requirements. The matrix template is created automatically using the information from the **Blocks** and **Requirements** worksheets.



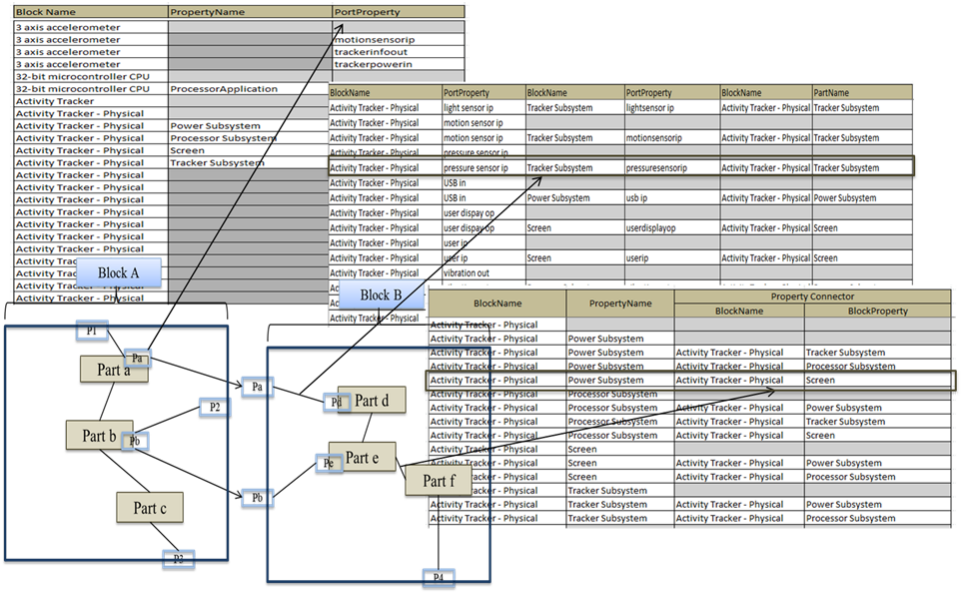
		Name	SmartPhone	WaterProof	Message Notification	Ease of Use	Alarm Notification	Style	Safety	Accurate	Battery Life	Store Data	Activity	Track Sleep Cycle	Track Steps	Heart Rate Monitor	Calorie count	Time	Selection	Display options
6	Blocks	3 axis accelerometer																		
7		32-bit microcontroller CPU																		
8		Activity Tracker																		
9		Activity Tracker - Physical																		
10		Ambient Light Sensor																		
11		Android																		
12		Barometeric Pressure Sensor																		
13		Battery																		
14		Bluetooth IC																		
15		Capacitive touch Screen																		
16		Galvanic Skin Response Sensor																		
17		GPS																		
18		Gym																		
19		IOS																		
20		Mission Context																		
21		Optical Heart Rate Monitor																		
22		PCB board																		
23		Physical Environment																		
24		Power Management Unit																		
25		Power Subsystem																		
26		Processor Subsystem																		
27		ProcessorApplication																		
28		Screen																		
29		Smart Phone																		
30		Storage unit																		
31		Tracker Subsystem																		
32		User																		
33		Vibration Motor																		
34		Watch																		
35		Water																		
36		Wireless Chipset																		

To create a satisfy relation between the blocks and requirements, identify the block that satisfies a requirement and in their intersection of row and column, enter 'x' to indicate that the corresponding requirement has been met. This creates a satisfy relation between block and requirement.

2.5 Internal Blocks Table

In the previous sections the system of interest has been defined with operations, values, and by different parts of the system. In this section, we will define how these parts of the system and its properties, will interact with each other.

To define ports through which the system interacts with other parts and subsystems, we create ports to blocks and then represent how these ports are connected. As shown in the diagram below, we can represent the interaction of block properties using ports and connectors.



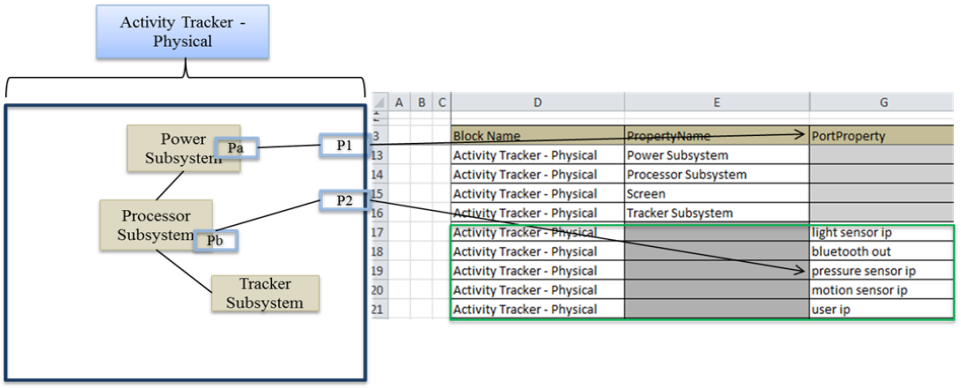
Block Property Table

This worksheet displays the blocks and their part properties based on how they are defined in previous worksheets. In addition to the part properties, you can create ports by using the **PortProperty** column.

Creating an entry is similar to entries discussed in other sections:

1. Specify the block to which a port has to be created
2. In the **PortProperty** column, enter a name for the port.

In the example below for the **Activity Tracker- Physical** block, the **PropertyName** column displays the existing part properties from previous worksheets. To create ports, enter the block name in the **Block Name** column (column D) and the port name in the **PortProperty** column (column G).



	A	B	C	D	E	G
3				Block Name	PropertyName	PortProperty
13				Activity Tracker - Physical	Power Subsystem	
14				Activity Tracker - Physical	Processor Subsystem	
15				Activity Tracker - Physical	Screen	
16				Activity Tracker - Physical	Tracker Subsystem	
17				Activity Tracker - Physical		light sensor ip
18				Activity Tracker - Physical		bluetooth out
19				Activity Tracker - Physical		pressure sensor ip
20				Activity Tracker - Physical		motion sensor ip
21				Activity Tracker - Physical		user ip

Block Connector Table

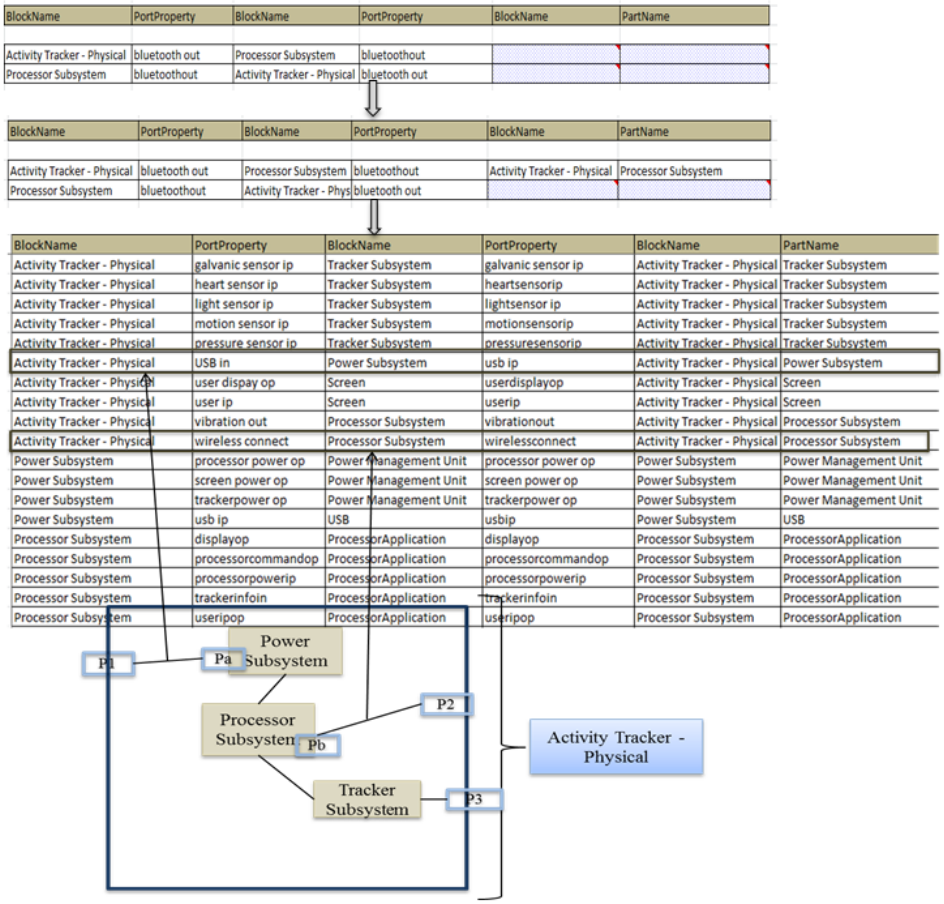
To create connectors between ports owned by the blocks and parts with ports, use the **BlockConnectorTable** worksheet. In the example below, the first two columns, **BlockName** and **PortProperty**, represent the owning block and the next two columns represent the connecting block port information.

The next columns, **BlockName** and **PartName**, are used to specify the owner and connecting parts.

To create the connection between ports, enter the top level block name and its port in the first two columns, followed by details of the connecting block and port. MapleMBSE will validate the input and create another row automatically to represent that the connection is bidirectional.

The last two columns will be highlighted. as shown below.

To complete the connection, enter the block name and part name in their respective columns, as shown below. The row that was automatically created by MapleMBSE does not require any input and should be left blank.



BlockName	PortProperty	BlockName	PortProperty	BlockName	PartName
Activity Tracker - Physical	bluetooth out	Processor Subsystem	bluetoothout		
Processor Subsystem	bluetoothout	Activity Tracker - Physical	bluetooth out		

BlockName	PortProperty	BlockName	PortProperty	BlockName	PartName
Activity Tracker - Physical	bluetooth out	Processor Subsystem	bluetoothout	Activity Tracker - Physical	Processor Subsystem
Processor Subsystem	bluetoothout	Activity Tracker - Phys	bluetooth out		

BlockName	PortProperty	BlockName	PortProperty	BlockName	PartName
Activity Tracker - Physical	galvanic sensor ip	Tracker Subsystem	galvanic sensor ip	Activity Tracker - Physical	Tracker Subsystem
Activity Tracker - Physical	heart sensor ip	Tracker Subsystem	heartsensorip	Activity Tracker - Physical	Tracker Subsystem
Activity Tracker - Physical	light sensor ip	Tracker Subsystem	lightsensor ip	Activity Tracker - Physical	Tracker Subsystem
Activity Tracker - Physical	motion sensor ip	Tracker Subsystem	motionsensorip	Activity Tracker - Physical	Tracker Subsystem
Activity Tracker - Physical	pressure sensor ip	Tracker Subsystem	pressuresensorip	Activity Tracker - Physical	Tracker Subsystem
Activity Tracker - Physical	USB in	Power Subsystem	usb ip	Activity Tracker - Physical	Power Subsystem
Activity Tracker - Physical	user dispay op	Screen	userdisplayop	Activity Tracker - Physical	Screen
Activity Tracker - Physical	user ip	Screen	userip	Activity Tracker - Physical	Screen
Activity Tracker - Physical	vibration out	Processor Subsystem	vibrationout	Activity Tracker - Physical	Processor Subsystem
Activity Tracker - Physical	wireless connect	Processor Subsystem	wirelessconnect	Activity Tracker - Physical	Processor Subsystem
Power Subsystem	processor power op	Power Management Unit	processor power op	Power Subsystem	Power Management Unit
Power Subsystem	screen power op	Power Management Unit	screen power op	Power Subsystem	Power Management Unit
Power Subsystem	trackerpower op	Power Management Unit	trackerpower op	Power Subsystem	Power Management Unit
Power Subsystem	usb ip	USB	usbip	Power Subsystem	USB
Processor Subsystem	displayop	ProcessorApplication	displayop	Processor Subsystem	ProcessorApplication
Processor Subsystem	processorcommandop	ProcessorApplication	processorcommandop	Processor Subsystem	ProcessorApplication
Processor Subsystem	processorpowerip	ProcessorApplication	processorpowerip	Processor Subsystem	ProcessorApplication
Processor Subsystem	trackerinfoin	ProcessorApplication	trackerinfoin	Processor Subsystem	ProcessorApplication
Processor Subsystem	useripop	ProcessorApplication	useripop	Processor Subsystem	ProcessorApplication

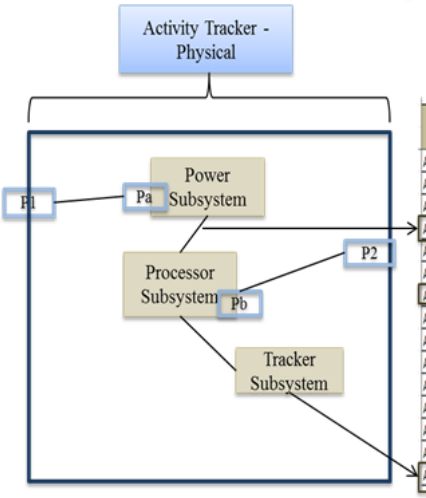
Property Connector Table

The **Property Connector Table** is used to connect part properties within the block, as shown below.

To create a connector between **Power Subsystem** and **Processor Subsystem**, as shown below, enter the block which owns the property in the first column. In this case, enter **Activity Tracker-Physical** followed by the owning part name, **Power Subsystem**. In the **Property Connector** section, enter the other property owner, which is again, the **Activity Tracker-Physical** and the connecting part, **Processor Subsystem**. When the entry is valid, MapleMBSE adds another row automatically to represent that the connection is bidirectional.

BlockName	PropertyName	Property Connector	
		BlockName	BlockProperty
Activity Tracker			
Activity Tracker - Physical			
Activity Tracker - Physical	Power Subsystem		
Activity Tracker - Physical	Processor Subsystem		
Activity Tracker - Physical	Power Subsystem		

BlockName	PropertyName	BlockProperty	Property Connector	
			BlockName	BlockProperty
Activity Tracker				
Activity Tracker - Physical				
Activity Tracker - Physical	Power Subsystem	Power Subsystem		
Activity Tracker - Physical	Processor Subsystem	Processor Subsystem		
Activity Tracker - Physical	Power Subsystem	Power Subsystem	Activity Tracker - Physical	Processor Subsystem
Activity Tracker - Physical	Processor Subsystem	Processor Subsystem	Activity Tracker - Physical	Power Subsystem



BlockName	PropertyName	Property Connector	
		BlockName	BlockProperty
Activity Tracker - Physical			
Activity Tracker - Physical	Power Subsystem		
Activity Tracker - Physical	Power Subsystem	Activity Tracker - Physical	Tracker Subsystem
Activity Tracker - Physical	Power Subsystem	Activity Tracker - Physical	Processor Subsystem
Activity Tracker - Physical	Power Subsystem	Activity Tracker - Physical	Screen
Activity Tracker - Physical	Processor Subsystem		
Activity Tracker - Physical	Processor Subsystem	Activity Tracker - Physical	Power Subsystem
Activity Tracker - Physical	Processor Subsystem	Activity Tracker - Physical	Tracker Subsystem
Activity Tracker - Physical	Processor Subsystem	Activity Tracker - Physical	Screen
Activity Tracker - Physical	Screen		
Activity Tracker - Physical	Screen	Activity Tracker - Physical	Power Subsystem
Activity Tracker - Physical	Screen	Activity Tracker - Physical	Processor Subsystem
Activity Tracker - Physical	Tracker Subsystem		
Activity Tracker - Physical	Tracker Subsystem	Activity Tracker - Physical	Power Subsystem
Activity Tracker - Physical	Tracker Subsystem	Activity Tracker - Physical	Processor Subsystem

2.6 Activity Diagram

An Activity Diagram is used to define system behavior. The top level system functionality is initially defined and these defined actions are further decomposed to show the logical behavior of the system.

Only call behavior actions with pins are used in this model.

Creating Actions for an Activity

Using the **Use Case** diagram, we have identified that the basic use case for the model is to track daily activity.

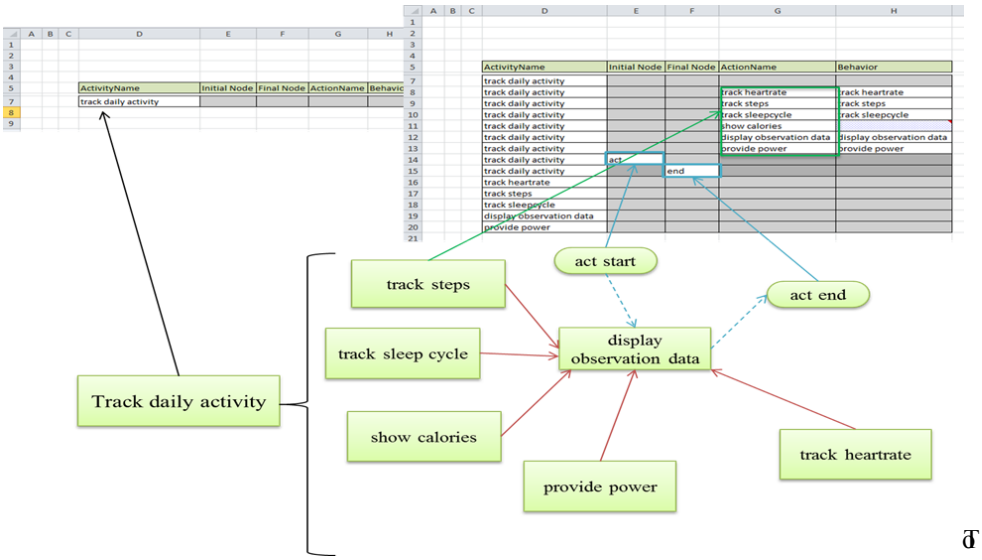
The **ActivityTable** worksheet is used to create activities, action, control flow and object flows. To create an activity diagram named, t**rack daily activity**, enter the name in the **ActivityName** column, as shown below. Once we create the actions for the activity, we need to now create flow between the actions. In this model, control flows are used only to represent the start and end of an activity.

To create control flows to denote the start and end of the activity, use the **ControlFlowName** column. Following the creation of the control flows, object flows can be created in the same worksheet.

Enter a name for the object flow in the **ObjectFlowName** column. As shown in the diagram below, control flows and object flows are created for the activity diagram, **track daily activity**. Linking these flows with actions is discussed in the following section.

Creating Actions for an Activity

Using the Use Case diagram we have identified the basic use case for our model is to track daily activity. The **ActivitysTable** worksheet is used to create activities, action, control flow and object



create an Activity Diagram, first create an activity and its elements in **ActivityTable**. Enter

the name of the activity in column D (**ActivityName**) as shown above. Once an activity is created we can create its initial node, final node and its actions in the respective column as shown. Use column H (**Behavior**) to allocate a behavior to the actions we created. In order to allocate a behavior, it should exist as an activity in the ActivityName Column.

Creating Flows

Using the ObjectFlow and ControlFlow Table we can now complete the activity diagram. To create an object flow between two actions in the ObjectFlow Table, Enter the activity under which the action was created in ActivityTable under Column D and the action name in Column E(Action Name Column) and the other action to be link with in Column G and its activity in Column F. Now MapleMBSE will create the input and output pins for the respective actions. In the case of Behavior being allocated to the action being links, MapleMBSE will automatically create parameters.

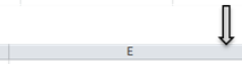
			Object Flow	
Activity Name	Action Name	Activity Name	Action Name	
track daily activity				
track daily activity	track heartrate			
track daily activity	track steps			
track daily activity	track sleepcycle			
track daily activity	show calories			
track daily activity	display observation data			
track daily activity	act			
track daily activity	end			
track daily activity	provide power			

		Object Flow	
Activity Name	Action Name	Activity Name	Action Name
track daily activity			
track daily activity	track heartrate		
track daily activity	track steps		
track daily activity	track sleepcycle		
track daily activity	show calories		
track daily activity	display observation data		
track daily activity	act		
track daily activity	end		
track daily activity	provide power		
track daily activity	track heartrate	track daily activity	display observation data

		Object Flow	
Activity Name	Action Name	Activity Name	Action Name
track daily activity			
track daily activity	track heartrate		
track daily activity	track steps		
track daily activity	track sleepcycle		
track daily activity	show calories		
track daily activity	display observation data		
track daily activity	act		
track daily activity	end		
track daily activity	provide power		
track daily activity	track heartrate	track daily activity	display observation data
track daily activity	track steps	track daily activity	display observation data
track daily activity	track sleepcycle	track daily activity	display observation data
track daily activity	show calories	track daily activity	display observation data
track daily activity	provide power	track daily activity	display observation data

To create control flow between nodes we follow the same steps we used for creating object flow, In ControlFlow Table, enter the activity name in Column C(ActivityName) and the node in Column D(Activity Node) and the action node to be linked with in Column F and its activity in Column E. Similarly we can link nodes and actions with Control or object flows.

	Activity Name	Swim Lane	Representing Block	ActivityPartition Allocation	
				Activity Name	Action Name
6	connect smartphone				
7	display notification				
8	display observation data				
9	display time				
10	provide power				
11	track daily activity				
12	track heartrate				
13	track sleepcycle				
14	track steps				



	Activity Name	Action Name	Control Flow	
			Activity Name	Action Name
6	track daily activity			
7	track daily activity	track heartrate		
8	track daily activity	track steps		
9	track daily activity	track sleepcycle		
10	track daily activity	show calories		
11	track daily activity	display observation data		
12	track daily activity	act		
13	track daily activity	end		
14	track daily activity	provide power		
15	track daily activity	act	track daily activity	display observation data
16	track daily activity	display observation data	track daily activity	end

To create an object flow between parameter and an action we use the same method we used for creating object flow between actions, MapleMBSE will automatically identify the element type and create the corresponding links.

Allocate Actions to Swim Lanes

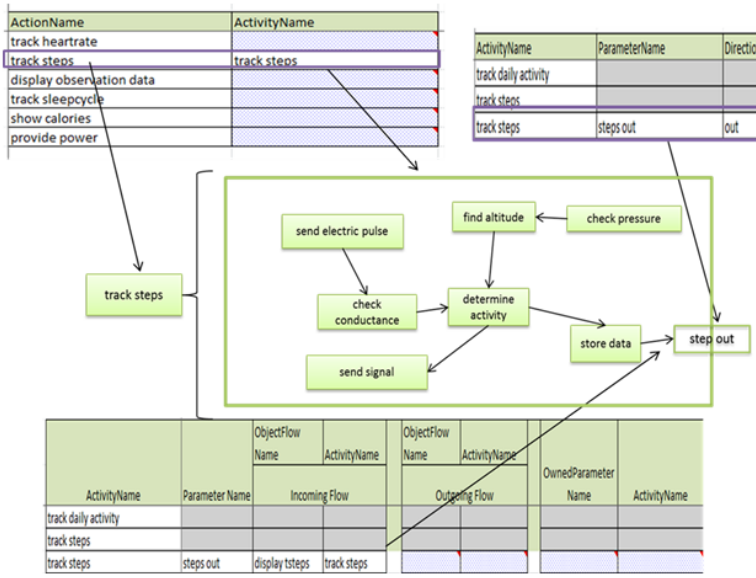
			ActivityPartition Allocation	
Activity Name	Swim Lane	Representing Block	Activity Name	Action Name
connect smartphone				
display notification				
display observation data				
display time				
provide power				
track daily activity				
track heartrate				
track sleepcycle				
track steps				
display observation data	PA	ProcessorApplication		

			ActivityPartition Allocation	
Activity Name	Swim Lane	Representing Block	Activity Name	Action Name
connect smartphone				
display notification				
display observation data				
display time				
provide power				
track daily activity				
track heartrate				
track sleepcycle				
track steps				
display observation data	PA	ProcessorApplication		
display observation data	PA	ProcessorApplication	display observation data	get sleep rate

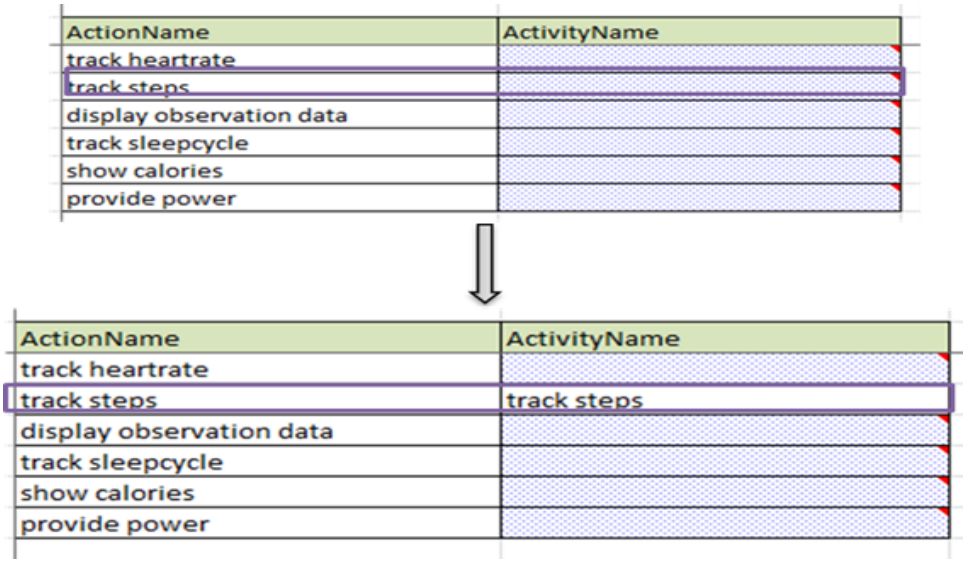
Before we can allocate actions to a swim lane. We first create swim lanes and assign a block to the swim lane. BlocksTable tab displays the list of available blocks that can be assigned to a swim lane. In SwimLanesTable tab ActivityName column displays the activities created using previous tables. To create a swim lane enter a name in Column G (Swim Lane) and the block it represents in Column H(Representing Block). Now we have created the swim lanes with the respective blocks it represents. To allocate an action enter the name of the activity in column F(Activity Name) and the swim lane name in column G. MapleMBSE will highlight this record as duplicate field, enter the action to be allocated in column J and the activity in column I to complete the allocation. MapleMBSE will now accept this as a valid record and remove the error. Similar we can access different activities we created and allocate the actions to swim lanes.

Activity Breakdown

To create an activity diagram for an action, use the **ActivityBreakdown** worksheet. An action can be further decomposed to more detailed granularity by using the same approaches discussed in previous sections.



To create an activity diagram for an action, enter the same name in the **ActivityName** column, as shown below. This will create an activity diagram, **track steps.** Using the **ActivityTable** worksheet, we can create flows and actions with the same approach mentioned above.



Creating Activity Parameters

When we create activity diagrams for the actions, consistency with the number of input pins and output pins must be maintained. We create a parameter and specify the direction, as shown below.

Enter a name in the **ParameterName** column and its **Direction** as either **in** or **out**. We create the parameter to show that the **track steps** activity receives information that is processed by actions within **track steps** and output is sent through a parameter called **steps out**.

ActivityName	ParameterName	Direction
track daily activity		
track steps		
track steps	steps out	out

Parameter Flows

The **ParameterFlowTable** worksheet is similar to that of the **ObjectFlowTable** worksheet. The only difference is that we link object flows with an owned parameter of the activity instead of action pins. Once the activity parameter is created using the **Parameters** worksheet, we create an object flow in the **ActivityTable** worksheet. The object flow created is then linked with an action using the object flow table. In the parameter flow table, enter the activity followed by the parameter, **steps out**, in the **Parameter Name** column. MapleMBSE will highlight the row. Enter the object flow name, **display tsteps**. At this point, we should also specify that the parameter used, **steps out**, is an owned parameter for the activity, as shown below.

		ObjectFlow		ObjectFlow			
ActivityName	Parameter Name	Name	ActivityName	Name	ActivityName	OwnedParameter Name	ActivityName
		Incoming Flow		Outgoing Flow			
track steps							
track steps	steps out	display tsteps	track steps			steps out	track steps

3 State Machine Diagram

This section defines how to create states, define their transitions and the events that trigger these transitions using MapleMBSE. The configuration file, **TWCSysML-StateMachines.MSE** defines four worksheets that can be used to create states and define their transitions. The **TWCSysML-StateMachines.MSE** file is located in the **Application** subdirectory of your MapleMBSE installation directory. This example only covers the case where a transition is triggered by a signal event. The following package structure is followed:

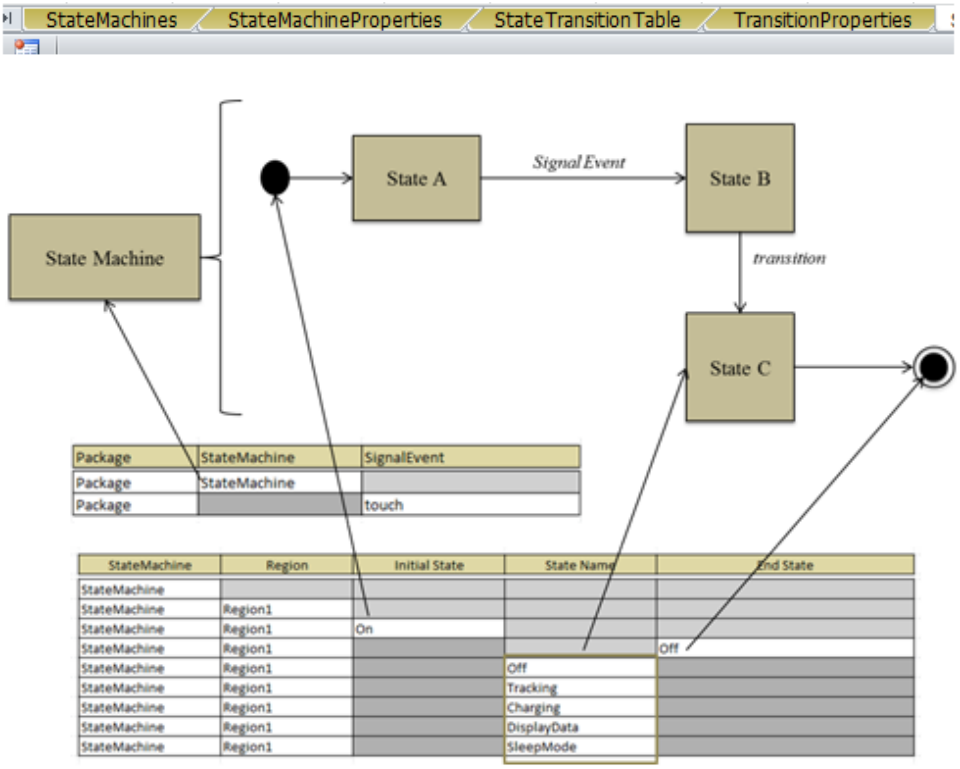
- Model

-Package

+StateMachine

+Region

+SignalEvent



3.1 How to Create a State Machine Diagram

In the **StateMachines** worksheet enter **Package** name as **Package**, and **StateMachine** name as **StateMachine**. These naming conventions can be changed by modifying the configuration file.

Package	StateMachine	SignalEvent
Package	StateMachine	
Package		touch

Once the state machine is created, we have to define a region in which states will be created. To create a region, use the **Pseudo State Properties** worksheet. Enter a name for the region, as shown in the table below (**Region1** is used as default as defined in configuration file). This table is also used to create the pseudo state (**PseudoState** column) and final state (**FinalState** column) that defines the start and end of the state machine. Enter a name for the states, as shown below. We define the transition from the pseudo state in this worksheet, once we have created other states, and its transition in the **Transition Matrix Table** worksheet.

StateMachine	Region	Pseudo State	Final State	Source State	Transition	Target State
StateMachine						
StateMachine	Region1					

StateMachine	Region	Pseudo State	Final State	Source State	Transition	Target State
StateMachine						
StateMachine	Region1					
StateMachine	Region1		Off			
StateMachine	Region1	On				

3.2 How to Create States and Transitions

To create a transition between states in the **State Transition Table**, enter the source state in the **SourceState** column and the target in the **TargetState** column, as shown below. Once we create these transitions between the states, we can edit the properties of these transitions in **TransitionProperties** worksheet.

SourceState	TargetState
Charging	
DisplayData	
Off	
On	
SleepMode	
Tracking	

SourceState	TargetState
Charging	
DisplayData	
Off	
On	
SleepMode	
Tracking	
On	Tracking
Tracking	DisplayData
Tracking	Charging
Charging	SleepMode
Tracking	Off

3.3 How to Create Triggers with Signal Events

Initially, the **Transition Name** column will be displayed as a blank column since we haven't named the transitions. Enter a name for the transitions so they can be identified to create a trigger and assign a signal event. Enter the Transition name in the **Transition Name** column followed by the **Signal Event** name we created in **StateMachine Table**. MapleMBSE will accept this as a valid input and automatically populate the other fields.

Transition Name	Source State	Target State	Signal Event
	Tracking	Off	
	On	Tracking	
	Tracking	DisplayData	
	Tracking	Charging	
	Charging	SleepMode	

Transition Name	Source State	Target State	Signal Event
usercommand/power	Tracking	Off	
powerOn	On	Tracking	
touch	Tracking	DisplayData	
usb connect	Tracking	Charging	
Charge/Non-Tracking	Charging	SleepMode	

Transition Name	Source State	Target State	Signal Event
usercommand/power	Tracking	Off	
powerOn	On	Tracking	
touch	Tracking	DisplayData	
usb connect	Tracking	Charging	
Charge/Non-Tracking	Charging	SleepMode	
touch	Tracking	DisplayData	touch
usb connect	Tracking	Charging	usb connect

4 Count Down Timer Model

The example is create with the following package structure

Model

-Requirements

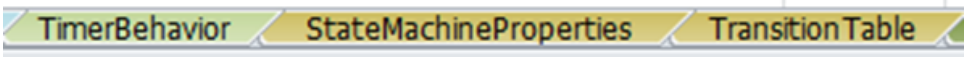
-Use Case

-Timer

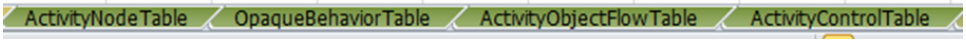
To create a Timer model we define the simplest requirements that is expected of the Timer. The model is required to have functions that enable the user to start, reset, pause and stop the timer. When Timer reaches zero, the user must be notified and the timer should continue counting down. Keeping these as the only requirements, a Requirements table is initially created. From these requirements we identify the actors and use cases. We create a Timer block to define its behavior based on these identified Use Cases.

To define the Timer properties, we create operations and properties to the Timer block. To enable the user to reset, stop, pause etc., we create these as signals so the user can command the system when it is being executed. **State Machine** and **Activities** are used to define the system behavior and its different states of operation.

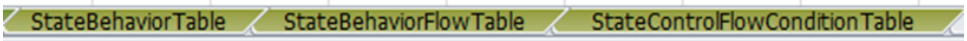
The **RequirementsTree** and **UseCases** worksheets are used to define the requirements that should be met by the model and its use cases. The **CountDownTimer**, **SignalTable**, and **TimeEventTable** worksheets are used to create blocks and events that will trigger the system to transition to a different state.



The **TimerBehavior** worksheet is used to create a *StateMachine* that will define the states at which the system will exist and its behavior at different states. It is also used to create operations and activities that will define system behavior. The **StateMachineProperties** worksheet is used to create the states and the **TransitionTable** worksheet is used to define their transition and events that triggered them.



ActivityNode Table and OpaqueBehavior Table is used to create activity nodes and behaviors. ActivityObjectFlow table and ActivityControlTable is used to create flows between the actions and nodes created in previous tables.



The **StateBehaviorTable**, **StateBehaviorFlowTable** and **StateControlFlowCondtionTable** worksheets are similar to that of previously mentioned worksheets. The only difference being that they are used to create activity flows that define states entry behavior.

4.1 Requirements Table

The **Requirements Table** worksheet is used to create *Requirements*. The configuration file is defined in a way that this table can be use to create two levels of requirements. As shown below, requirements for the system are created.

RequirementsSatisfy Table worksheet is used to create a *Satisfy* relation between the *Requirements*, *Blocks* and its properties. This table will be used to verify if the requirements are met once the system has been created.

Requirements Table

Requirement		Requirement 2nd Level		
ID*	**Name**	**ID***	**Name**	**Specification**
1	Timer			
1	Timer	1.1	Accurate	The timer should count down every 1 second.
1	Timer	1.2	Functions	The timer must have functions to start, reset, pause and notify user.
1	Timer	1.3	Working	The timer should continue counting even after 0, until user signals to stop.
1	Timer	1.4	Notify	The timer should notify the user at 0.

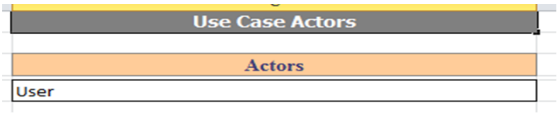
4.2 UseCase Table

The **Actors** tab is used to identify the actors, while the **UseCases** tab is used to associate these Actors with UseCases.

To create an association between Actor and UseCase, enter the Actor Name in the **Actor** column, followed by the UseCase in **UseCase1** column.

To create an association between UseCases, Enter the **Actor** name in **Actor** column followed by the UseCase name in the **UseCase1** column and associating UseCase in the **Associated-UseCase2** column.

The **UseCases** table is created as shown below :



Use Case Actors
Actors
User

UseCases			
Actor	UseCase1	Associated UseCase2	Associated UseCase3
User	count down		
User	count down	notified	
User	count down	notified	count down
User	count down	pause	
User	count down	pause	count down
User	count down	reset	
User	count down	reset	count down
User	notified		
User	notified	count down	
User	notified	count down	notified
User	notified	count down	pause
User	notified	count down	reset
User	pause		
User	pause	count down	
User	pause	count down	notified
User	pause	count down	pause
User	pause	count down	reset
User	reset		
User	reset	count down	
User	reset	count down	notified
User	reset	count down	pause
User	reset	count down	reset

4.3 CountDownTimer Table

This table is used to create the Timer block, signals & events that will be used later in creating the model.

To create a block, enter the name in the **Block Name** column.

To create signals, enter a name for the signal in the **Signals** column, and its package name in the **PackageName** column

Note: Two kinds of events can be created in this worksheet, Signal events and Time events. These events are created based on the signals that are being used.

Timer Events & Signals					
Package Name	**Block Name**	**Signals**	**Signal Events**	**Timed Event**	**Instances**
CountDownTimer					
CountDownTimer	Timer				
CountDownTimer					instance
CountDownTimer		reset			
CountDownTimer		notified			
CountDownTimer		timeup			
CountDownTimer		start			
CountDownTimer		pause			
CountDownTimer		stop			
CountDownTimer		resume			
CountDownTimer			startEvent		
CountDownTimer			stopEventA		
CountDownTimer			pauseEvent		
CountDownTimer			resumeEvent		
CountDownTimer			stopEventB		
CountDownTimer			stopEvent		
CountDownTimer			resetEvent		
CountDownTimer			notifyEvent		
CountDownTimer			timeupEvent		
CountDownTimer				TimeEvent	

Signal Table

The **Signal** table is an extension of the previous section. Here, we relate the signals that were created with the SignalEvent. Later in the model, we will use these signal events as triggers to define transition between states.

To assign a signal to *SignalEvent, e*nter the *SignalEvent* name from the previous table and its corresponding signal in the **Signals** column.

Signal Event	
SignalEvent	**Signals**
notifyEvent	
timeupEvent	
startEvent	
stopEventA	
pauseEvent	
resumeEvent	
stopEventB	
stopEvent	
resetEvent	

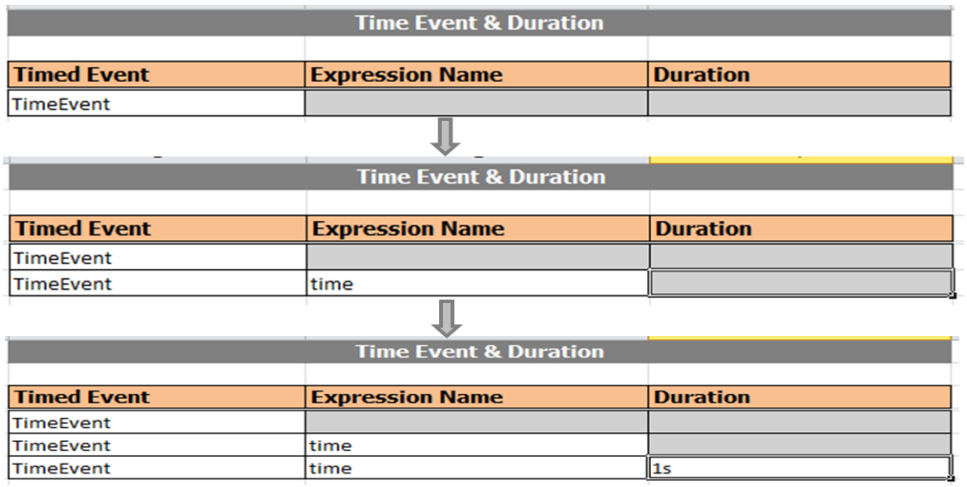
Signal Event	
SignalEvent	**Signals**
notifyEvent	notified
timeupEvent	
startEvent	
stopEventA	
pauseEvent	
resumeEvent	
stopEventB	
stopEvent	
resetEvent	

Time Event Table

The **Time Event** table is used to create the duration for the timed event.

Enter the event name in the **Timed Event** column, followed by a name for the duration in the **Expression Name** column.

Next, enter the required time duration in the **Duration** column. Assign the duration to the *TimeEvent* by entering the event and expression name in their respective columns.



Time Event & Duration		
Timed Event	**Expression Name**	**Duration**
TimeEvent		

Time Event & Duration		
Timed Event	**Expression Name**	**Duration**
TimeEvent		
TimeEvent	time	

Time Event & Duration		
Timed Event	**Expression Name**	**Duration**
TimeEvent		
TimeEvent	time	
TimeEvent	time	1s

Now we have created the necessary E*vents* and *Signal* that will be used to define the *State* and *Transition* for the system.

4.4 Timer Behavior Table

Using the Timer Behavior table, we will define properties, operations and the behavior aspect of the system using *State Machines* and *Activities*.

To create a property, enter the block in the **Block Name** column and its property in the **Block Property** column.

Based on the use case, we will create the operations expected of the system: *restart*, *count-down* and *notify*.

To create operations, enter a name for the operation and the block in the respective columns.

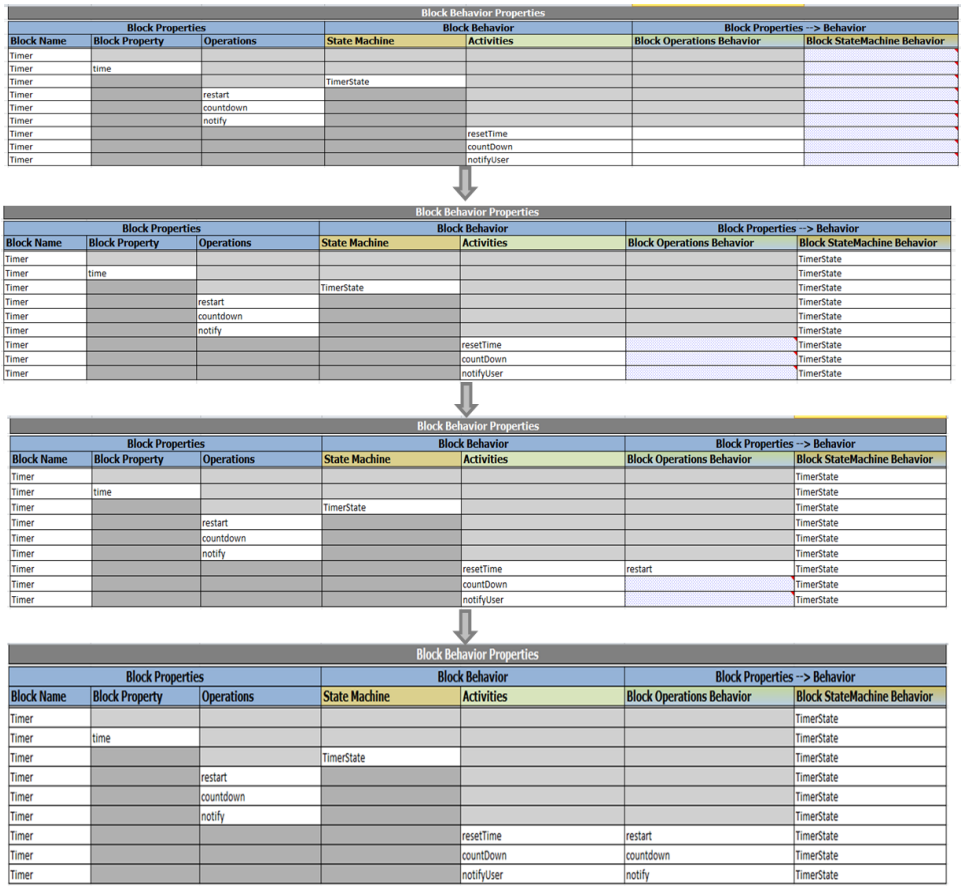
Next, we will create a *StateMachine* to define the system.

Enter the block name in the **Block Name** column and, in the same row, enter a name for the *StateMachine* in the **StateMachine** column. This will create a *StateMachine* for the Timer Block as shown below.

To make sure that the Timer Block exhibits the behavior of the *StateMachine* entered in the previous step, enter the *StateMachine* in the **Block StateMachine Behavior** column. In doing this, we are defining the state machine as a classified behavior.

Next, we create activities based on the operations created for the block.

Enter the block name in the **Block Name** column and the activity name in the **Activities** column, we have now created activities for the block Timer. In the **Block Operations Behavior** column, enter the respective operations for the activities created.



Block Behavior Properties

Block Properties			Block Behavior		Block Properties --> Behavior	
Block Name	Block Property	Operations	State Machine	Activities	Block Operations Behavior	Block StateMachine Behavior
Timer						
Timer	time					
Timer			TimerState			
Timer		restart				
Timer		countdown				
Timer		notify				
Timer				resetTime		
Timer				countDown		
Timer				notifyUser		

Block Behavior Properties

Block Properties			Block Behavior		Block Properties --> Behavior	
Block Name	Block Property	Operations	State Machine	Activities	Block Operations Behavior	Block StateMachine Behavior
Timer						TimerState
Timer	time					TimerState
Timer			TimerState			TimerState
Timer		restart				TimerState
Timer		countdown				TimerState
Timer		notify				TimerState
Timer				resetTime		TimerState
Timer				countDown		TimerState
Timer				notifyUser		TimerState

Block Behavior Properties

Block Properties			Block Behavior		Block Properties --> Behavior	
Block Name	Block Property	Operations	State Machine	Activities	Block Operations Behavior	Block StateMachine Behavior
Timer						TimerState
Timer	time					TimerState
Timer			TimerState			TimerState
Timer		restart				TimerState
Timer		countdown				TimerState
Timer		notify				TimerState
Timer				resetTime	restart	TimerState
Timer				countDown		TimerState
Timer				notifyUser		TimerState

Block Behavior Properties

Block Properties			Block Behavior		Block Properties --> Behavior	
Block Name	Block Property	Operations	State Machine	Activities	Block Operations Behavior	Block StateMachine Behavior
Timer						TimerState
Timer	time					TimerState
Timer			TimerState			TimerState
Timer		restart				TimerState
Timer		countdown				TimerState
Timer		notify				TimerState
Timer				resetTime	restart	TimerState
Timer				countDown	countdown	TimerState
Timer				notifyUser	notify	TimerState

4.5 StateMachine Properties Table

Next, we define the states and region for the *TimerState* we created previously.

Enter the *StateMachine* name followed by the region name in the **Region** column.

Create the Initial and Final states and the states at which the system will exist in respective columns, as shown below.

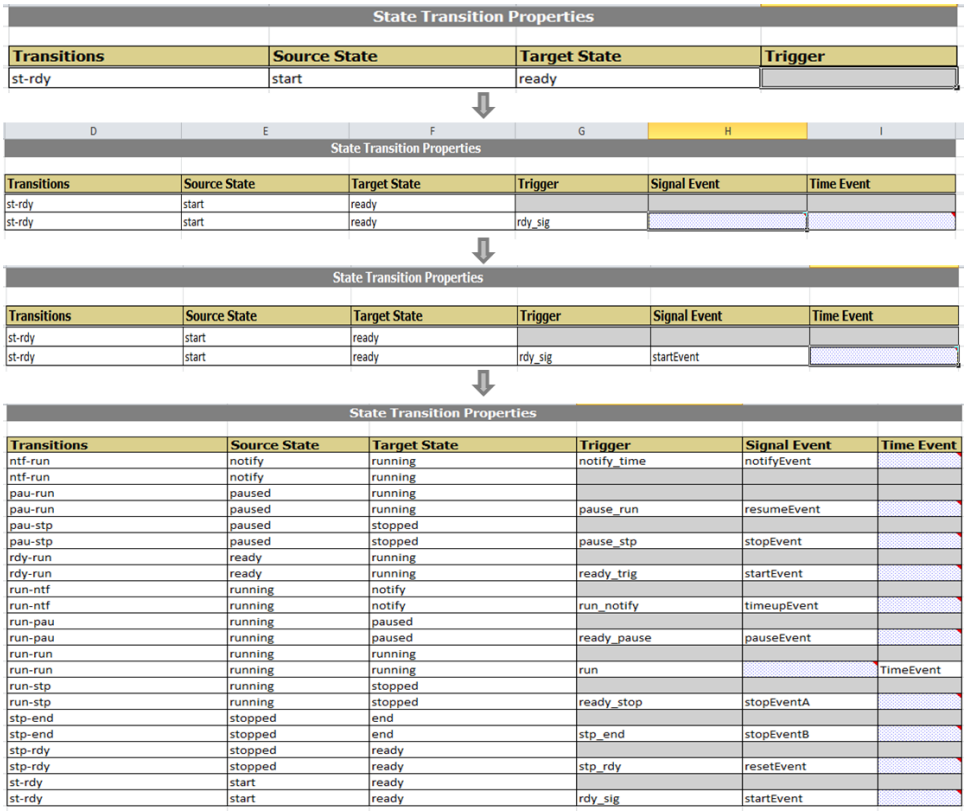
StateMachine Properties				
State Machine	**Region**	**Initial State**	**States**	**Final State**
TimerState				
TimerState	Region			
TimerState	Region	start		
TimerState	Region			end
TimerState	Region		end	
TimerState	Region		notify	
TimerState	Region		paused	
TimerState	Region		ready	
TimerState	Region		running	
TimerState	Region		stopped	

Transition Table

To create a transition between states with triggers, enter a name for the transition in the **Transitions** column (a row will be added with Source and Target state cells highlighted).

Enter the source state in the **Source State** column and the target state in the **Target State** column to create a transition between them.

To add a trigger that starts the transition, enter the transition name and trigger name. The source and target state fields will be updated automatically. To add an event to the trigger, enter the event name in the appropropriate column. For example, to assign *startEvent* as a trigger between the start and ready states, enter the transition name, then provide a name for the trigger. Since startEvent is a signal event, it is populated in the **Signal Event** column, as shown.



State Transition Properties

Transitions	Source State	Target State	Trigger
st-rdy	start	ready	

D	E	F	G	H	I

State Transition Properties

Transitions	Source State	Target State	Trigger	Signal Event	Time Event
st-rdy	start	ready			
st-rdy	start	ready	rdy_sig		

State Transition Properties

Transitions	Source State	Target State	Trigger	Signal Event	Time Event
st-rdy	start	ready			
st-rdy	start	ready	rdy_sig	startEvent	

State Transition Properties

Transitions	Source State	Target State	Trigger	Signal Event	Time Event
ntf-run	notify	running	notify_time	notifyEvent	
ntf-run	notify	running			
pau-run	paused	running			
pau-run	paused	running	pause_run	resumeEvent	
pau-stp	paused	stopped			
pau-stp	paused	stopped	pause_stp	stopEvent	
rdy-run	ready	running			
rdy-run	ready	running	ready_trig	startEvent	
run-ntf	running	notify			
run-ntf	running	notify	run_notify	timeupEvent	
run-pau	running	paused			
run-pau	running	paused	ready_pause	pauseEvent	
run-run	running	running			
run-run	running	running	run		TimeEvent
run-stp	running	stopped			
run-stp	running	stopped	ready_stop	stopEventA	
stp-end	stopped	end			
stp-end	stopped	end	stp_end	stopEventB	
stp-rdy	stopped	ready			
stp-rdy	stopped	ready	stp_rdy	resetEvent	
st-rdy	start	ready			
st-rdy	start	ready	rdy_sig	startEvent	

4.6 ActivityNodeTable

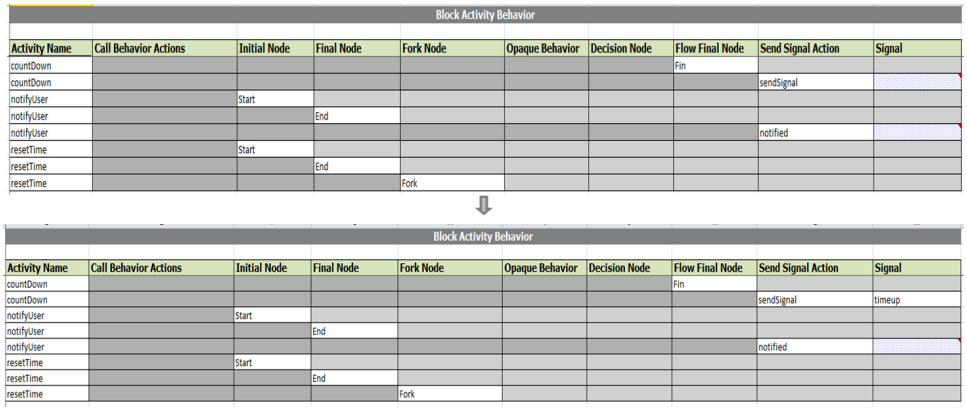
Next, we define the activity created in the **TimerBehavior** table,

To create actions and flow for an activity, enter the name of activity to which the above mentioned elements will be created.

In the **Call Behavior Actions** column, enter a name to create call behavior actions.

Similarly, this table is used to create initial an final nodes, forks, opaque behaviors, decision nodes, and send signal actions. Each of which can be created by providing a name for the node and its activity.

To assign the signal that will be send when a signal action is invoked, enter the name in the **Send Signal Action** column and the signal that will be sent in the **Signal** column (signals that were created in *CountDownTimer* table).



Block Activity Behavior

Activity Name	Call Behavior Actions	Initial Node	Final Node	Fork Node	Opaque Behavior	Decision Node	Flow Final Node	Send Signal Action	Signal
countDown							Fin		
countDown								sendSignal	
notifyUser		Start							
notifyUser			End						
notifyUser								notified	
resetTime		Start							
resetTime			End						
resetTime				Fork					

Block Activity Behavior

Activity Name	Call Behavior Actions	Initial Node	Final Node	Fork Node	Opaque Behavior	Decision Node	Flow Final Node	Send Signal Action	Signal
countDown							Fin		
countDown								sendSignal	timeup
notifyUser		Start							
notifyUser			End						
notifyUser								notified	
resetTime		Start							
resetTime			End						
resetTime				Fork					

Opaque Behavior Table

This sheet is used to assign *OpaqueBehavior* to an action and define its parameter and equation.

To assign *OpaqueBehavior* to an action, enter the Opaque Behavior created in previous table in the **Opaque Behavior** column.

Note: The available actions will be automatically listed in **Opaque Action** column, as shown below.

To create an equation, enter it in the **Opaque Equation** column.

Opaque Behavior Properties			
Opaque Behavior Name	Parameters	Direction	Opaque Equation
Opq_behavior			

Opaque Behavior Properties			
Opaque Behavior Name	Parameters	Direction	Opaque Equation
Opq_behavior			time_out=t_in-1

To manipulate the parameters and direction, we first need to create links between the actions.

Activity ObjectFlow Table

This table is used to create object flow between activities.

To create object flow between actions, enter the source action name in column E (**Activity Node** column) and its activity in the **ActivityName** column followed by the target action information in column G(**ActivityNode** column) and its activity in the **Activity Name** column.

The object flows between the actions are created, as shown below.

ObjectFlow Table			
Activity Name	**Activity Node**	**Activity Name**	**Activity Node**
countDown			
countDown	get_updatedValue	countDown	Fork
countDown	Fork	countDown	read_time
countDown	Fork	countDown	update_time
countDown	read_time	countDown	ForkN
countDown	evaluate_Expression	countDown	update_time
countDown	ForkN	countDown	evaluate_Expression
countDown	ForkN	countDown	Decision
notifyUser			
resetTime			
resetTime	Fork	resetTime	update_newValue
resetTime	Fork	resetTime	new_Input
resetTime	get_oldValue	resetTime	reset_oldValue
resetTime	reset_oldValue	resetTime	update_newValue
resetTime	new_Input	resetTime	update_newValue
resetTime	reset_toZero	resetTime	reset_oldValue
resetTime	get_newValue	resetTime	Fork

Activity ControlFlow Table

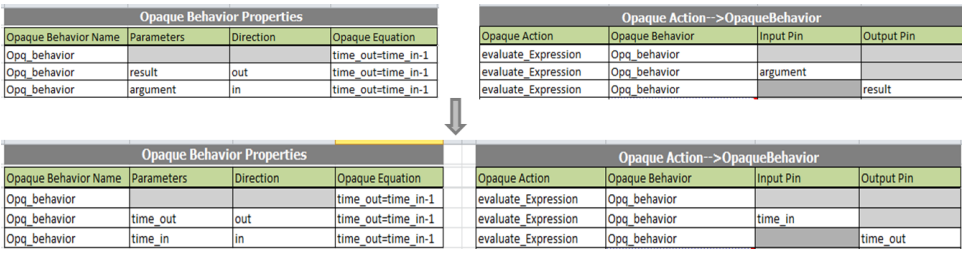
The **Activity Control Flow** table works similar to the **Object Flow** table,

Enter the source action and activity name in the first two columns, followed by the target activity and action name.

ControlFlow Table			
Activity Name	**Activity Node**	**Activity Name**	**Activity Node**
countDown			
countDown	get_updatedValue		
countDown	Fork		
countDown	Start		
countDown	Start	countDown	read_time
countDown	End		
countDown	read_time		
countDown	read_time	countDown	evaluate_Expression
countDown	update_time		
countDown	update_time	countDown	Decision
countDown	evaluate_Expression		
countDown	evaluate_Expression	countDown	update_time
countDown	ForkN		
countDown	Decision		
countDown	Decision	countDown	Fin
countDown	Decision	countDown	sendSignal
countDown	Fin		
countDown	sendSignal		
countDown	sendSignal	countDown	End
notifyUser			
notifyUser	sendNotification		
notifyUser	sendNotification	notifyUser	notified
notifyUser	Start		
notifyUser	Start	notifyUser	sendNotification
notifyUser	End		
notifyUser	notified		
notifyUser	notified	notifyUser	End
resetTime			
resetTime	Fork		
resetTime	Start		
resetTime	Start	resetTime	reset_oldValue
resetTime	End		
resetTime	get_oldValue		
resetTime	reset_oldValue		
resetTime	reset_oldValue	resetTime	update_newValue
resetTime	new_Input		
resetTime	reset_toZero		
resetTime	get_newValue		
resetTime	update_newValue		
resetTime	update_newValue	resetTime	End

Once we have completed the Behavior flow tables, we have to sync the input and output flow of Opaque Behavior and its call action. To do this, go back to the **Opaque Behavior** table.

The Input and Output pins will be displayed as argument and result by default. We change this value based on the Opaque Equation parameter. Rename the argument in both tables to *time_in* and *time_out* instead of *result* and *argument* for the Opq_behavior.



Opaque Behavior Properties			
Opaque Behavior Name	Parameters	Direction	Opaque Equation
Opq_behavior			time_out=time_in-1
Opq_behavior	result	out	time_out=time_in-1
Opq_behavior	argument	in	time_out=time_in-1

Opaque Action-->OpaqueBehavior			
Opaque Action	Opaque Behavior	Input Pin	Output Pin
evaluate_Expression	Opq_behavior		
evaluate_Expression	Opq_behavior	argument	
evaluate_Expression	Opq_behavior		result

Opaque Behavior Properties			
Opaque Behavior Name	Parameters	Direction	Opaque Equation
Opq_behavior			time_out=time_in-1
Opq_behavior	time_out	out	time_out=time_in-1
Opq_behavior	time_in	in	time_out=time_in-1

Opaque Action-->OpaqueBehavior			
Opaque Action	Opaque Behavior	Input Pin	Output Pin
evaluate_Expression	Opq_behavior		
evaluate_Expression	Opq_behavior	time_in	
evaluate_Expression	Opq_behavior		time_out

We have created state machines and activities to define the behavior of the system. As of now *StateMachine* and the activities are defined as seperate behaviors of the same system. In the following section, we will define how the system behaves at each state using the activities we created.

4.7 State Behavior Table

The State Behavior table will list the states created in the **StateMachine Properties** worksheet.

Next, we will assign an entry behavior to the system.

In the example, we will create an entry behavior to the running state. Enter the state name in the **State Name** column.

In the **State Entry Behavior** column, enter a name to create an entry behavior (*decrease* in this example).

Next, we will define nodes and actions to the entry behavior, as shown below.

To assign a behavior to the call actions we created in an earlier section, enter the behavior you want to assign in the **Behavior** column adjacent to the call actions.

State Entry Behavior Table					
State Name	State Entry Behavioir	Initial Node	Final Node	Call Behavior Actions	Behavior
end					
notify					
paused					
ready					
running					
stopped					

State Entry Behavior Table					
State Name	State Entry Behavioir	Initial Node	Final Node	Call Behavior Actions	Behavior
end					
notify					
paused					
ready					
running					
stopped					
running	decrease				

State Entry Behavior Table					
State Name	State Entry Behavioir	Initial Node	Final Node	Call Behavior Actions	Behavior
end					
notify					
paused					
ready					
running					
stopped					
running	decrease				
running	decrease	start			
running	decrease		end		
running	decrease			decrease	

State Entry Behavior Table					
State Name	State Entry Behavioir	Initial Node	Final Node	Call Behavior Actions	Behavior
end					
notify					
paused					
ready					
running					
stopped					
running	decrease				
running	decrease	start			
running	decrease		end		
running	decrease			decrease	countDown

State Behavior ControlFlow Table

Creating behavior control flows is similar to creating activity control flows.

Enter the source action and activity in the first two columns and target action and activity in the next column.

State Behavior ControlFlow Table			
State Activity	**State Activity Node**	**State Activity**	**State Activity Node**
decrease			
decrease	end		
decrease	start		
decrease	start	decrease	decrease
decrease	decrease		
decrease	decrease	decrease	end
notify			
notify	start		
notify	start	notify	notify
notify	end		
notify	notify		
notify	notify	notify	end
reset			
reset	start		
reset	start	reset	reset
reset	end		
reset	reset		
reset	reset	reset	end
test			

We have now created the control flows. When we defined a requirement initially, we stated that the system should notify the user when time reaches zero and should continue counting down even after reaching zero. To achive this, we will set a guard condition to the control flow of the merge node created in earlier sections. In a previous section, we have already create a notify behavior to the state and to send a signal to user.

State ControlFlow Condition Table

In the **ControlFlow Condition** table, existing contol flows will be listed based on previous inputs.

To create a guard condition, enter the state activity name in the **State Activity** column followed by the source and target activity node information and enter a guard condition.

State Behavior ControlFlow Condition Table			
State Activity	**State Activity Node (Source)**	**State Activity Node (Target)**	**Control Guard Condition**
resetTime			
countDown			
notifyUser			
countDown	Decision	Fin	
countDown	Decision	sendSignal	
countDown	evaluate_Expression	update_time	
countDown	read_time	evaluate_Expression	
countDown	sendSignal	End	
countDown	Start	read_time	
countDown	update_time	Decision	
notifyUser	notified	End	
notifyUser	sendNotification	notified	
notifyUser	Start	sendNotification	
resetTime	reset_oldValue	update_newValue	
resetTime	Start	reset_oldValue	
resetTime	update_newValue	End	

State Behavior ControlFlow Condition Table			
State Activity	**State Activity Node (Source)**	**State Activity Node (Target)**	**Control Guard Condition**
countDown			
countDown	Start	read_time	
countDown	read_time	evaluate_Expression	
countDown	update_time	Decision	
countDown	evaluate_Expression	update_time	
countDown	Decision	Fin	
countDown	Decision	sendSignal	
countDown	sendSignal	End	
notifyUser			
notifyUser	sendNotification	notified	
notifyUser	Start	sendNotification	
notifyUser	notified	End	
resetTime			
resetTime	Start	reset_oldValue	
resetTime	reset_oldValue	update_newValue	
resetTime	update_newValue	End	
countDown	Decision	Fin	time>0||time<0

State Behavior ControlFlow Condition Table			
State Activity	**State Activity Node (Source)**	**State Activity Node (Target)**	**Control Guard Condition**
countDown			
countDown	Start	read_time	
countDown	read_time	evaluate_Expression	
countDown	update_time	Decision	
countDown	evaluate_Expression	update_time	
countDown	Decision	Fin	
countDown	Decision	sendSignal	
countDown	sendSignal	End	
notifyUser			
notifyUser	sendNotification	notified	
notifyUser	Start	sendNotification	
notifyUser	notified	End	
resetTime			
resetTime	Start	reset_oldValue	
resetTime	reset_oldValue	update_newValue	
resetTime	update_newValue	End	
countDown	Decision	Fin	time>0||time<0

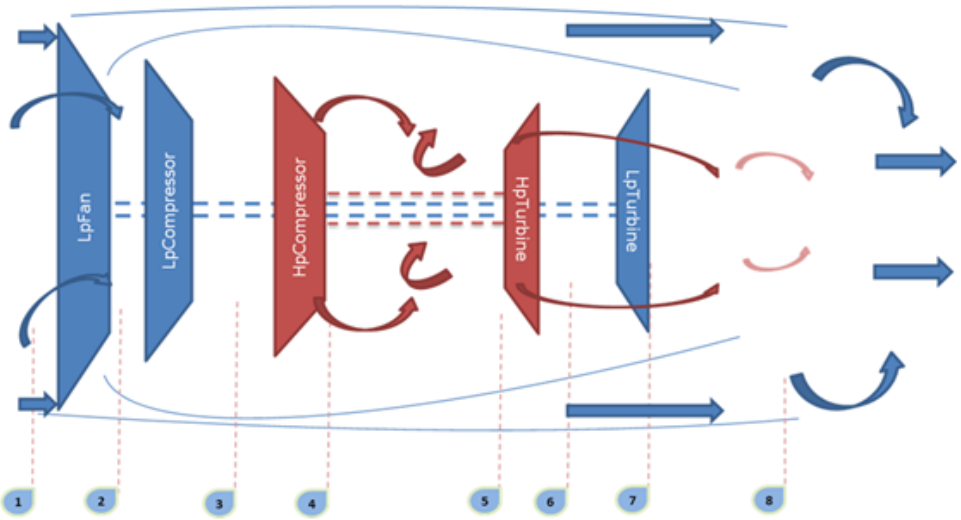
State Behavior ControlFlow Condition Table			
State Activity	**State Activity Node (Source)**	**State Activity Node (Target)**	**Control Guard Condition**
countDown			
countDown	Start	read_time	
countDown	read_time	evaluate_Expression	
countDown	update_time	Decision	
countDown	evaluate_Expression	update_time	
countDown	Decision	Fin	
countDown	Decision	Fin	time>0||time<0
countDown	Decision	sendSignal	
countDown	Decision	sendSignal	time=0
countDown	sendSignal	End	
notifyUser			
notifyUser	sendNotification	notified	
notifyUser	Start	sendNotification	
notifyUser	notified	End	
resetTime			
resetTime	Start	reset_oldValue	
resetTime	reset_oldValue	update_newValue	
resetTime	update_newValue	End	

5 Turbofan Engine Model

5.1 Introduction

This example model is used to identify design points of a turbofan engine. MapleMBSE and Cameo Systems Modeler™ were used to create a turbofan example model. The design point calculations are based on ideal gas turbine cycle analysis.
Initially, a mission statement is defined to specify the scope of the model and to identify design points at Mach number 0.8 and operating altitude between 35000ft to 45000ft with a bypass ratio between 6-8.

5.2 Turbofan Model



The turbofan system is defined as shown in the diagram above. The system consists of a twin-spool configuration, with a high pressure turbine driving a high pressure compressor, a low pressure turbine driving a low pressure compressor, and a fan. Temperature and pressure are identified at the design points, as shown in the figure. The primary goal is to identify the design points with optimum SFC (specific fuel consumption) value.

5.3 Requirements

Once the mission statement is defined, system requirements for the turbofan are also stated for each subcomponent in terms of target efficiency, pressure ratio etc., which have to be satisfied. The **SystemRequirements** worksheet in MapleMBSE is used to define the specifications and target values that have to be achieved. In addition to the system specifications,

analysis requirements are created to define the input values which will be used to analyze the model.

To maintain traceability between system level requirements and mission level requirements, the **DeriveRequirements** worksheet in MapleMBSE is used to create derived relationships between requirements.

5.4 ValueType

The **ValueTypesTable** and **UnitQuantityKindTable** worksheets are used to define units and type of values that will be used to define the system. These valuetypes are used to specify the type of value properties of the system to be modeled.

5.5 Constraint Blocks

Constraint blocks are created and constraints that will be used in the system are captured using the **ConstraintProperties** worksheet. Similar to value types, these blocks are used to specify the type the constraint property of the system that will be defined.

5.6 System Model

The Turbofan Blackbox is used to specify the properties of the turbofan in terms of values, subcomponents and ports through with the system will interact.

Once the subcomponents are created we now define the values and constraint properties, then type them to valuetypes and the constraint block created. A specific worksheet view is created in MapleMBSE to show components values, constraints and their types.

An Analysis block is created to provide value exchange between the subcomponents. The Analysis block provides the default values with which the analysis is performed and also receives the results of analysis.

5.7 Results

The **InstanceResults** table is used to display the results of analysis performed in the model using simulation toolkit in Cameo Systems modeller. In MapleMBSE the results are mapped to Excel graph for visualization. This results worksheet is treated as read-only and used to only visualize the results of analysis at different altitudes.

To create a new instance:

1. Create a new instance specification by providing a name in the Instance Specification column in **InstanceTable** worksheet and type "Analysis Block" as the name of the block in the Instance of Block column.
2. Define the feature and corresponding value with which the new analysis has to be performed, required input values to be created are *ByPassRatioA* and *targetEfficiency_hpTurbine*.
3. Once the analysis block is defined, specify the inlet properties by creating a new instance for the InletConditions block, similar to the above method. The required values in this case are Ta(inlet static temperature in K) and Pa (inlet static pressure in bar).
4. Commit the changes to Teamwork Cloud.
5. Open the model in Cameo or Magic Draw, then create a new block diagram in the NewInstance package, drag and drop the analysis block instance.
6. Drop the inletConditions instance into the analysis instance to create a new feature instance for the Analysis block.
7. Right-click the analysis instance and select simulate to run the analysis.
8. Export the results of analysis as new instance into the Result package under NewInstance then commit to Teamwork Cloud.
9. Reload MapleMBSE to see the results in the **NewInstanceResults** worksheet.

To maintain the traceability between the requirements and the modeled system modeled, use **VerifyRequirementsMatrix** to have a verify relationship between system requirements and value properties of the block. By creating this verify relation, now we have traceability from system values to system requirements and from system requirements to mission requirements.

The **RequirementsTraceability** worksheet displays all the requirements from the model and its relationships such as trace, verify, derived with other model elements.

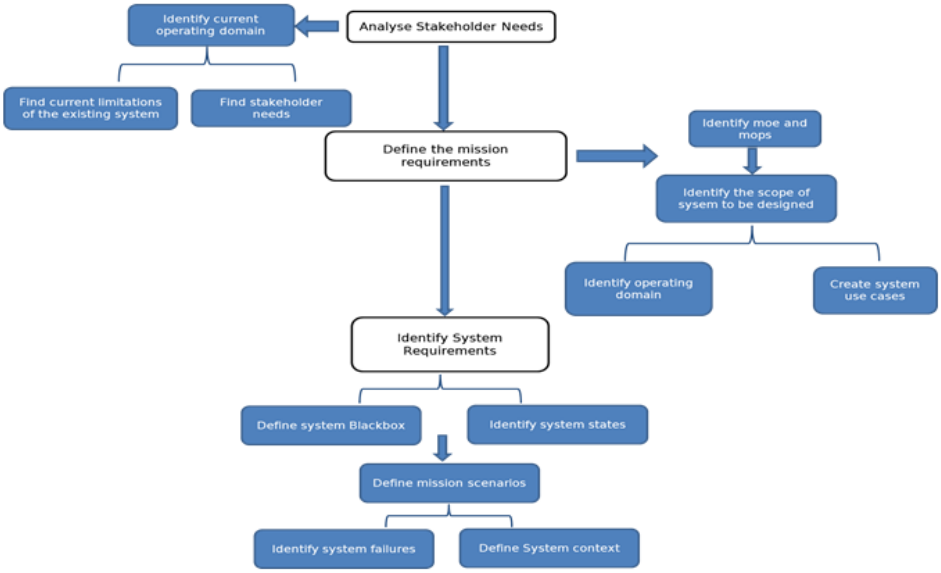
5.8 References

1. Cohen, H. Rogers. G. and Saravanamuttoo, H. (1996). Gas turbine theory. Harlow: Pearson education.
2. Sanford Friedenthal. (2015). A Practical Guide to SysML, 3rd Edition. Morgan Kaufmann Publishers.

6 UAV Model

6.1 Introduction

This model uses the Object Oriented System Engineering Method (OOSEM) to design a conceptual model of an Unmanned Aerial Vehicle (UAV). The primary use of UAV in consideration is to assist forest fire fighting operations in remote areas. The sample model shows a part of the OOSEM workflow to identify system requirements.



6.2 Analyze Stakeholder Needs

To identify the needs of stakeholders, in this case the fire department, the current operating domain is modeled to find the existing limitations and expectations of the fire department. The existing domain is captured using the block definition diagram represented in a table format in the **OperatingDomain** worksheet. A causal analysis is performed to identify the factors that are of interest to the fire department operation [6]. This causal analysis also reveals the present limitations in the fire department operation. At this stage, we have identified the needs of stakeholder based on which we will derive the mission requirements.

6.3 Mission Requirement

To determine the scope and mission of the UAV model, we first identify the measure of effectiveness based on the stakeholder needs analysis. Secondly, we define the operating domain in which the system to be modeled will operate. The operating domain is represented using a block diagram and shown in table format using the **OperatingDomainUAV** worksheet. We identify the use cases to determine the high level behavior of the system and its interaction. Next, from the measure of effectiveness and the operating domain, we can define the Mission Requirements and stakeholder requirements from the stakeholder needs that we identified.

6.4 System Requirements

Before identifying the system requirements, we define units, and interfaces that will be used by the system of interest. A separate package called Interface is create using the **InterfaceTable** to contain the flows and signals that will be used in the model.

System Behavior

To find the system requirements, we initially define the UAV blackbox that displays: ports through which the system interacts, its parts, and its values. In addition, we also define the operations that are expected of the system, and the method to achieve it in terms of activities. The **UAVBlackBox** worksheet displays the model elements mentioned above. Now we define the system behavior and represent states at which the system will operate and its events. On identifying the mission profile of UAV, we create detailed states at which the system should operate. Following this, we use activities to define system behavior. Based on the use cases, we create the activities since our mission is to control forest fires and we are still in the conceptual phase. We define system behavior based on this activity.

Weight Estimation

Once we have defined the system behavior we need to determine the system specification in order to create the system requirements. To identify the general design requirements the weight of the UAV is first estimated followed by sizing and identifying critical parameters. The **WeightEstimationTable** worksheet displays the value properties and constraint properties need to estimate the weight of UAV. This worksheet also has tables created in excel that displays specifications of similar aircraft and estimation constants from historical data [1]. Based on the mission profile the parameter values can be altered based on payload, range, endurance, etc. when satisfactory values are determined the values are updated to WeightEstimationBlock and saved to the model in Teamwork Cloud.

Wing Area Estimation

To determine the sizing we initially create the constraints using the **WingAreaConstraint** worksheet. Similar to the weight estimation worksheet, the **WingAreaEstimation** worksheet is used to find wing area by iterating key parameters. Using the matching plot technique [2] Wing loading vs Thrust loading is plotted from which we identify the wing area.
We have estimated the weight and wing area based on which other design parameters can be further evaluated. This example model covers the conceptual phase from stakeholder need analysis to identify system requirements.

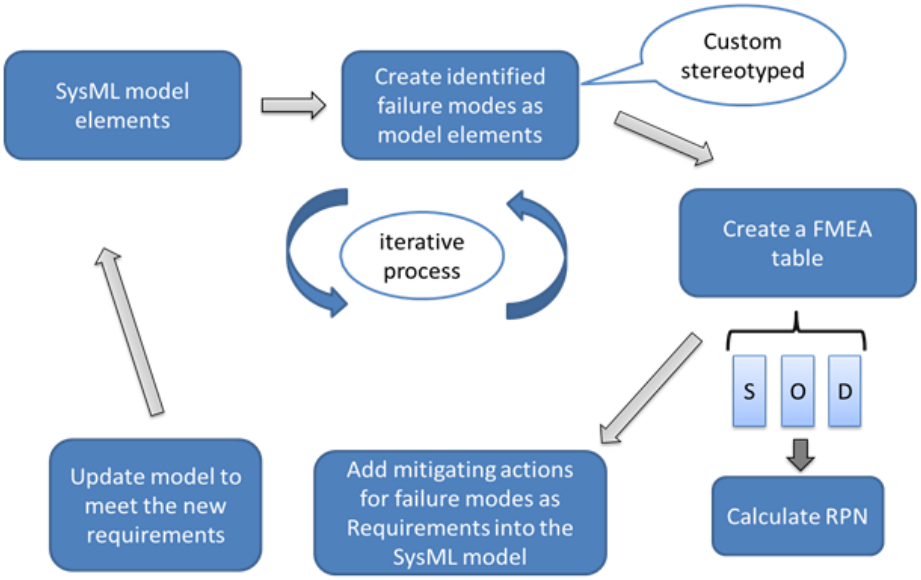
6.5 References

1. Austin, R. (2010). Unmanned air vehicles: UAVS design, development, and deployment. Chichester, West Sussex, and U.K.: Wiley.
2. Raymer, D. P. (1992). Aircraft design: A conceptual approach. Washington, D.C.: AIAA.
3. Sadraey, M. H. (2017). Unmanned aircraft design: A review of fundamentals. San Rafael, CA: Morgan & Claypool.
4. Sadraey, M. H. (2013). Aircraft design: A systems engineering approach. Hoboken, NJ: Wiley.
5. Simard, A. J., & Forster, R. B. (1972). A survey of air tankers and their use. Ottawa: Forest Fire Research Institute.
6. Sanford Friedenthal. (2015). A Practical Guide to SysML, 3rd Edition. Morgan Kaufmann Publishers.
7. GLOBAL HAWK SYSTEMS ENGINEERING CASE STUDY.pdf. (n.d.). Retrieved from https://www.scribd.com/document/409826283/GLOBAL-HAWK-SYSTEMS-ENGINEERING-CASE-STUDY-pdf
8. Firefighting Aircraft Recognition Guide - California - PDF Free Download. (n.d.). Retrieved from https://docobook.com/-firefighting-aircraft-recognition-guide-california.html

7 FMEA Template

7.1 Introduction

This model is used to perform FMEA analysis by accessing SysML model elements from Teamwork Cloud server. This example shows a FMEA process to identify possible failure modes of system functions defined in conceptual design of a UAV; however this template can be used to perform FMEA on different model elements by specifying appropriate path and elements in the configuration file.



The FMEA process is performed as shown in the figure, system functions from the model are accessed and failure modes are identified. Further we identify severity, occurrence and detection for the failure modes and calculate the RPN (Risk Priority Number). Mitigating actions for identified failures are created as new requirements. The complete process is saved back to the teamwork cloud model.

7.2 FMEA

The **FMEAMatrix** worksheet is used to identify new failure modes for the system function and to create a dependency (identifiedFM). Once we create new failure modes, we use the **FMEATable** worksheet to provide a detailed analysis of the potential failure by specifying S, O and D from which RPN is calculated.

7.3 Recommended Action

In this process, recommended actions are captured as requirements that can be saved back to the model. The **RequirementFMEAMatrix** worksheet is used to create a custom dependency (deriveFMEA) between identified FMEA and recommended actions. The **FMEARequirementTable** worksheet is used to add specification to the new requirements created as a result of this analysis.

To use the custom FMEA template:

1. Add the TWCSysML.mdzip model to the teamwork cloud server.
2. In Cameo Systems Modeler or Magic Draw, Right-click CustomStereotypes profile→ Project Usage →Export Packages to New Server project.
3. In desired project File→ Project Usage →Server Project select the exported profile from previous step.
4. Update path in the MSE file to get model elements.

7.4 References

1. Kratzke, R. (2018). Failure Modes Effects Analysis in MBSE. [ebook] Available at: https://www.incose.org/docs/default-source/texas-gulf-coast/tgcc-conference-2018/2018-papers/kratzke-2018-incose-presentation-(for-public-distribution).pdf?sfvrsn=db4796c6_2 [Accessed 22 May 2019].
2. Publishing, R. (2019). Failure Mode and Effect Analysis - FMEA - and Criticality Analysis - FMECA. [online] Weibull.com. Available at: https://www.weibull.com/basics/fmea.htm [Accessed 22 May 2019].