intercax

75 Fifth Street NW, Suite 312
Atlanta, GA 30308, USA
Voice: +1- 404-592-6897
Web: www.InterCAX.com
E-mail: info@intercax.com

ParaMagic® 18.0
User Guide

Table of Contents

1 About **4**

2 New Features **7**

3 Quick Start **9**

3.1 First Pass – execute existing models *9*

3.2 Second Pass – create new models *10*

4 Installation **11**

4.1 Installation Requirements *11*

4.1.1 System Requirements 11

4.1.2 MagicDraw Requirements 11

4.1.3 Core Solver Requirements 12

4.2 Installation ProcessF *12*

5 User Documents and Models **17**

5.1 Users Guide *17*

5.2 Tutorials *17*

5.3 Examples *18*

5.4 Relevant Publications *19*

6 SysML Model Requirements **19**

6.1 Structural requirements *19*

6.1.1 Model schema requirements 19

6.1.2 Model instance requirements 20

6.2 Naming requirements *20*

6.3 Mathematical expression requirements *21*

6.4 Value types and QUDV profile *21*

6.5 Math constants and functions *22*

6.6 Conditional Functions and Operators *24*

6.7 Aggregate Properties and Functions *24*
6.7.1 Multiplicity 24
6.7.2 Instance Slot Values 24
6.7.3 Aggregate Functions and Operators 25
6.8 Complex Aggregate Relationships *25*
6.9 Recursion and Redefinition *29*
6.9.1 Recursion 29
6.9.2 Redefinition 32
6.10 Limitations *41*

7 Program Features 44
7.1 Command Menus *44*
7.2 Browser *46*
7.2.1 "Solution in progress" Window 47
7.2.2 Variable Browser 48
7.2.3 Toolbar 48
7.2.4 Relationship Browser 48
7.2.5 Editing an Instance in the Browser Window 49
7.3 ParaMagic® Settings *49*
7.4 ParaMagic® Working Folder *52*

8 Connections to External Tools 52
8.1 ParaMagic - Custom Mathematica Connection *52*
8.1.1 Installation 53
8.1.2 Usage 53
8.1.3 Graphing Functions 54
8.1.4 Statistical Functions 55
8.1.5 User-Defined Mathematical Functions 55
8.1.6 UserfnN.m 56
8.1.7 Custom Functions 56
8.2 ParaMagic - Excel Connection *56*
8.2.1 Generate/Update SysML Instance Model Structure from Excel Spreadsheets 56
8.2.2 Update SysML Instance Model Values from Excel Spreadsheets 66
8.3 ParaMagic - MATLAB Connection *75*
8.3.1 Using MATLAB scripts 81
8.3.2 Using MATLAB functions 83

9 Trade Studies 84
9.1 Operation *84*
9.2 Limitations *88*

10 ParaMagic® Silent (NEW) 89
10.1 Invoking ParaMagic® in silent mode from plugins *89*
10.1.1 Java API calls to invoke ParaMagic® Silent 89
10.1.2 Install, Test, and Review the ParaMagic Silent Demo plugin 93
10.2 Invoking ParaMagic® in silent mode from the ParaMagic® plugin *95*

Copyright © 2014 InterCAX LLC. All Rights Reserved.

11 Copyright 96

11.1 Copyright statement from InterCAX LLC 96

11.2 Liability disclaimer from InterCAX LLC 96

Copyright © 2014 InterCAX LLC. All Rights Reserved.

1 About

This is the User Guide for **ParaMagic® 18.0 plugin for MagicDraw/SysML 17.0.5 and above**. The sections in this document and their purpose are stated below:

- **New Features** in ParaMagic® 18.0
- Quick Start – speed up learning ParaMagic® (run existing models, create new models)
- Installation – installation instructions for ParaMagic®
- User Documents – important documents for using ParaMagic® (including tutorials and examples)
- SysML Model Requirements – requirements for creating SysML models executable in ParaMagic®
- Program Features – learn about the ParaMagic® plugin user interface
- Connections to External Solvers – learn about incorporating Mathematica functions, MATLAB functions/scripts & Simulink models, and Excel spreadsheets as part of SysML parameteric models, and using ParaMagic to execute parametric models that require multiple solvers
- Trade Studies – learn to setup and run trade studies using ParaMagic®
- ParaMagic® Silent – learn to run ParaMagic® in silent mode and invoke it in from your own plugins
- Updates – learn about key user-oriented updates since the last release
- Copyright –important copyright information that users must read

Copyright © 2014 InterCAX LLC. All Rights Reserved.

ParaMagic® comes in two editions—ParaMagic® Lite and ParaMagic®. The table below compares the features of these editions. All features (both editions) are described in this Users Guide. Features highlighted in **red** are new in this release.

Features	PM Lite	PM
1) **Regular math solving** *Library of math functions available*	**Yes**	**Yes**
2) **Use Mathematica / PlayerPro (bundled[1]) / OpenModelica (free) / MATLAB Symbolic Math Toolbox as a core solver**	**Yes**	**Yes**
3) **ParaMagic® Silent** *Java API methods to call ParaMagic® in silent mode from your plugins and scripts to validate, solve, and update SysML models with results*	**No**	**Yes**
4) **Read/Write SysML instance values from/to Excel spreadsheets** *Connect SysML instances to Excel spreadsheets and read/write from/to spreadsheets.*	**Yes**	**Yes**
5) **Generate & update SysML instance model from Excel spreadsheets** *Automatically generate and update SysML instance models from tables in Excel spreadsheets.*	**No**	**Yes**
6) **Trade Studies** *Execute parametric model to perform trade studies*	**Yes**	**Yes**
7) **MATLAB/Simulink Connection** *Wrap MATLAB functions/scripts as constraint blocks and use in parametric models*	**No**	**Yes**
8) **Custom Mathematica Connection** *Wrap custom Mathematica functions as constraint blocks and use in parametric models*	**No**	**Yes**
9) **Complex Aggregates (see section 6.8)** *Solve parametric models with complex aggregate relationships.*	**No**	**Yes**
10) **Recursion and Redefinition (see section 6.9)** *Solve parametric models with recursive structures and implicit/explicit redefinition of properties*	**Yes**	**Yes**

[1] ParaMagic® 18.0 comes bundled with PlayerPro math solver (Wolfram Research). Current users can upgrade to ParaMagic®+PlayerPro bundled installation for a one-time nominal fee.

The following conventions are followed in this document:

- ParaMagic® implies this ParaMagic® 18.0 plugin
- MagicDraw SysML implies MagicDraw UML + MagicDraw SysML plugin
- Text enclosed by < > implies that you need to provide your system-specific settings, such as installation directory path or IP address.
- Text written in this **font** implies that it is a computer keyword or an abbreviation for a computer keyword.
- MD implies MagicDraw
- <MD_Root> refers to the MagicDraw installation directory (e.g. **C:\Program Files\MagicDraw_UML_17.0.5** on a Windows machine).
- OM implies OpenModelica
- MATLAB SMT implies MATLAB Symbolic Math Toolbox[2]

MagicDraw[3] is a registered trademark of No Magic, Inc.

Mathematica[4] is a registered trademark of Wolfram Research, Inc.

MATLAB[5] and Simulink[6] are registered trademarks of The MathWorks, Inc.

OpenModelica[7] is a freely-available modeling and simulation tool as part of the OpenModelica Project supported by the Open Source Modelica Consortium (OSMC).

PlayerPro[8] is a registered trademark of Wolfram Research, Inc.

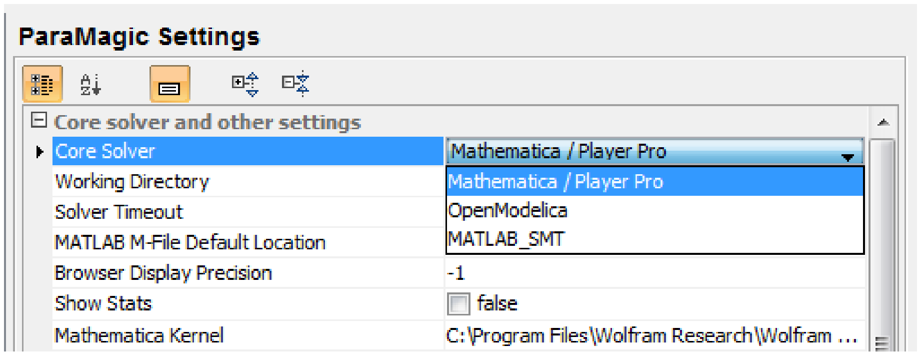
[2] MATLAB Symbolic Math Toolbox: http://www.mathworks.com/products/symbolic/
[3] MagicDraw: http://www.magicdraw.com
[4] Mathematica (Wolfram Research): http://www.wolfram.com/
[5] MATLAB (The MathWorks) - http://www.mathworks.com/products/matlab/
[6] Simulink (The MathWorks) - http://www.mathworks.com/products/simulink/
[7] OpenModelica - https://openmodelica.org/
[8] PlayerPro - http://www.wolfram.com/player-pro/

Copyright © 2014 InterCAX LLC. All Rights Reserved.

2 New Features

The following new features are introduced in ParaMagic® 18.0:

1. **Availability of ParaMagic® + Wolfram Player Pro bundle**
   With ParaMagic® 18.0, Wolfram Player Pro is available bundled with ParaMagic® at a very low incremental price. All of ParaMagic's advanced parametric solving capabilities are currently (and in future) available with Player Pro. The option to select Player Pro is now available in the ParaMagic Settings category under MagicDraw Environment Option (Options > Environment), as shown below. We highly recommend all our users who were not using Mathematica earlier to upgrade to this bundle to explore the full power of ParaMagic®.



ParaMagic Settings

Core solver and other settings

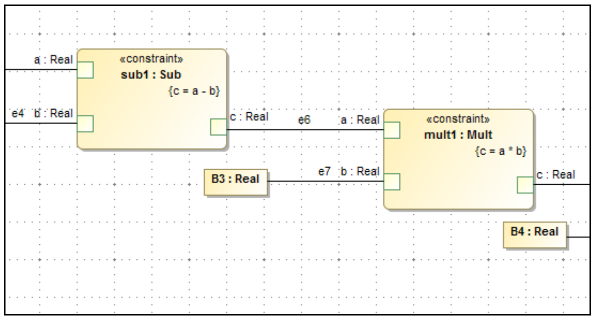
Core Solver	Mathematica / Player Pro
Working Directory	Mathematica / Player Pro
Solver Timeout	OpenModelica
MATLAB M-File Default Location	MATLAB_SMT
Browser Display Precision	-1
Show Stats	false
Mathematica Kernel	C:\Program Files\Wolfram Research\Wolfram ...

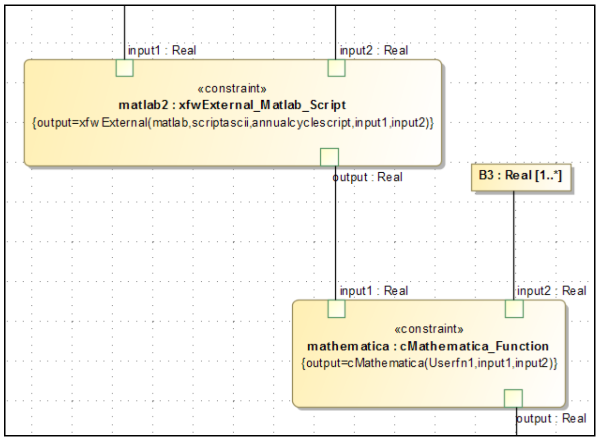
2. **Availability of ParaMagic® + Mathematica bundle**
   While Player Pro serves as an advanced math solver for ParaMagic®, users may still need the full power of Mathematica front end, such as the ability to work with Mathematica notebooks, advanced user interface for authoring custom Mathematica functions and scripts, access to Mathematica library and documentation, and the ability to test and debug Mathematica functions before using them in SysML parametric models. Users can also purchase Mathematica® bundled with ParaMagic® at a discounted incremental price.

3. **ParaMagic® Silent (Java API) - Invoke ParaMagic in silent mode from your own plugins**
   ParaMagic® 18.0 comes with the ParaMagic® Silent feature which provides two main capabilities for automation: (1) ParaMagic® Silent provides Java API methods to validate and solve parametric models, and update SysML instance models with results. Users who have installed the ParaMagic® plugin can now invoke these API methods from their own plugins; (2) ParaMagic® Silent also enables users to solve parametric models and update SysML instance models without launching the ParaMagic® browser interface. See section 10 for details on ParaMagic® Silent.

4. **Significant improvement in solving speed with PlayerPro and Mathematica**
   ParaMagic® 18.0 comes with a strong integration with Player Pro and Mathematica, communicating directly with the math kernel versus the file-based communication earlier. This offers significant improvements in solving speeds, especially for parametric trade studies executing 1000s of scenarios, and significant reduction in the communication overhead with Mathematica and PlayerPro solver engines.

5. **Support for binding connectors between constraint parameters**
   ParaMagic® 18.0 now supports binding connectors between two constraint parameters, as shown below. This works for all types of constraint blocks, those with regular math equations as well as those wrapping externally defined MATLAB functions/scripts and Mathematica functions, as shown below.



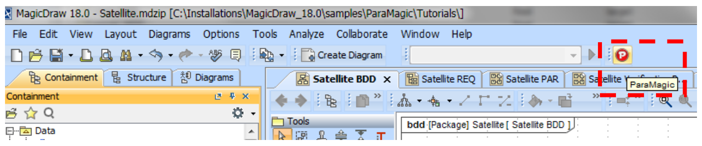

input1 : Real
input2 : Real
«constraint»
matlab2 : xfwExternal_Matlab_Script
{output=xfwExternal(matlab,scriptascii,annualcyclescript,input1,input2)}
output : Real
B3 : Real [1..*]
input1 : Real
input2 : Real
«constraint»
mathematica : cMathematica_Function
{output=cMathematica(Userfn1,input1,input2)}
output : Real

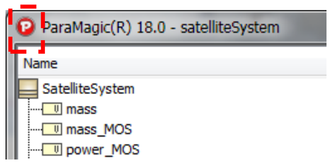
6. **Improved error handling and user messages when validating parametric models**
   ParaMagic® 18.0 comes with major improvements in handling errors and description of end user messages when validating parametric models.

7. **Improved support for MATLAB connection and messages**
   ParaMagic® 18.0 also provides major improvements in error handling and end user messages for the MATLAB/Simulink connection (section 8.3).

8. **Resolution of issues**
   In addition to the new and improved capabilities listed above, ParaMagic® 18.0 also resolves several issues, such as those related to: (1) reading/writing nested instance structure from/to Excel spreadsheets and (2) naming binding connectors and properties owned by blocks in read-only mode.

9. **New ParaMagic® Icon**
   ParaMagic® has a new icon . The new icon will show up in the ParaMagic® toolbar, browser, settings, and dialogs.


MagicDraw 18.0 - Satellite.mdzip [C:\Installations\MagicDraw_18.0\samples\ParaMagic\Tutorials\]
File Edit View Layout Diagrams Options Tools Analyze Collaborate Window Help
Create Diagram
Containment
Structure
Diagrams
Satellite BDD
Satellite REQ
Satellite PAR
Satellite
ParaMagic
Containment
Data
Tools
bdd [Package] Satellite [Satellite BDD]



3 QUICK START

3.1 First Pass – execute existing models

In this section, you will learn to execute two existing ParaMagic® models.

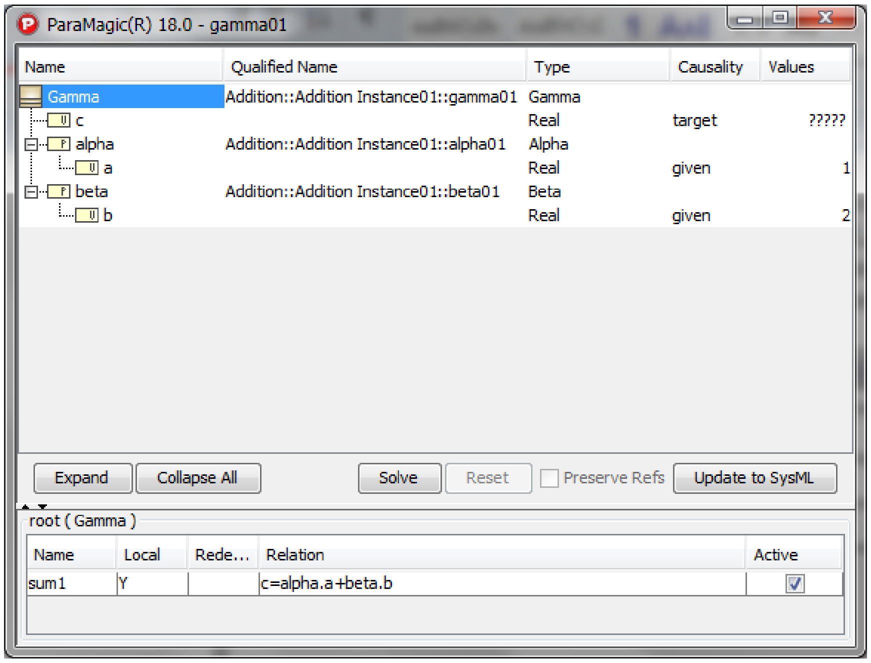
- The first model (Addition) is a basic model for testing ParaMagic® installation. In this model, a system variable is computed by adding two sub-system variables.

- The second model (Satellite) represents a simple satellite. It exercises basic SysML block definition and parametric modeling concepts and is thus useful for learning SysML parametrics.

Follow the steps below to get started.

1) Install ParaMagic® — see installation instructions in Section 3 below.

2) Open and execute the Addition model
   a) Start MagicDraw
   b) Open the Addition model located here:
      <MD_Root>\samples\ParaMagic\Tutorials\Addition.mdzip
   c) Locate the package **Addition::AdditionInstance01**.
   d) Right click on the package and select **ParaMagic > Browse**. This will launch the ParaMagic® browser, as shown in Figure 1.
   e) Click the **Solve** button. After successful solution, the value of **C** (target variable) should equal to 3.



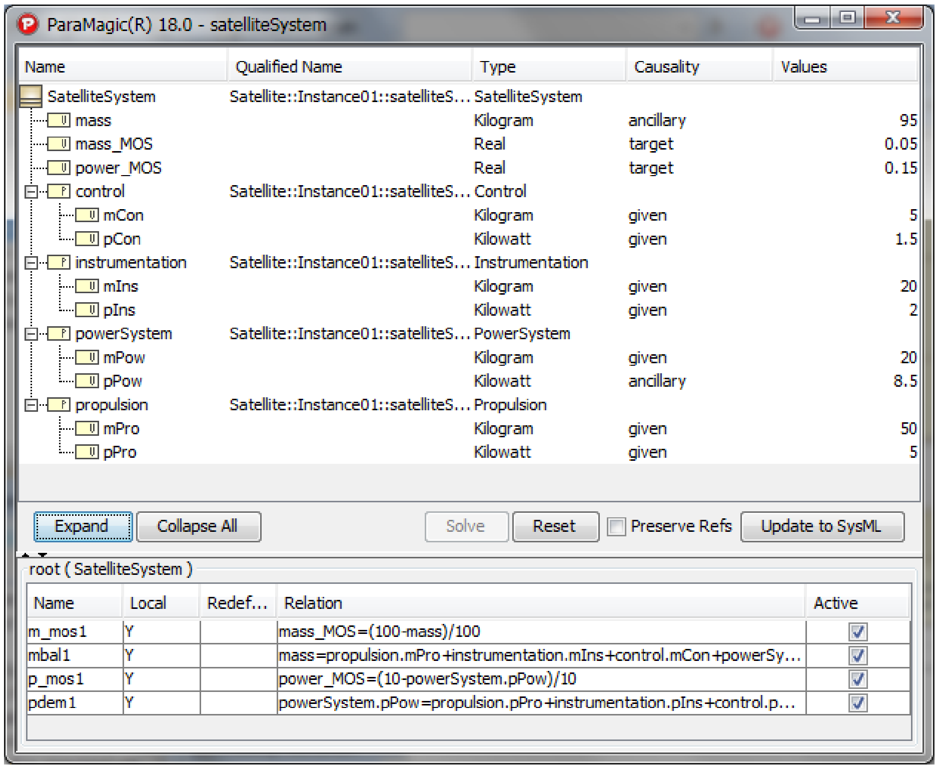
Name	Qualified Name	Type	Causality	Values
Gamma	Addition::Addition Instance01::gamma01	Gamma		
c		Real	target	?????
alpha	Addition::Addition Instance01::alpha01	Alpha		
a		Real	given	1
beta	Addition::Addition Instance01::beta01	Beta		
b		Real	given	2

Name	Local	Rede...	Relation	Active
sum1	Y		c=alpha.a+beta.b	✓

Figure 1: ParaMagic® browser showing Addition model instance (solved state)

Note: You should have configured a core solver during installation (Section 3). If you have not installed PlayerPro or Mathematica or OpenModelica (free) or MATLAB SMT, you can use our test Mathematica server for a trial period (30 days). For this, you will need an internet connection. It is possible that your enterprise firewall may block connection to our server.

3) Open and execute the Satellite model
   a) Open the Satellite model located here: **<MD_Root>\samples\ParaMagic\Tutorials\Satellite.mdzip**
   b) Locate the package **Satellite::Instance01**.
   c) Right click on the package and select **ParaMagic > Browse**.
   d) Click on the **Expand** button to expand the model.
   e) Click on the **Solve** button. After solving, the ParaMagic® browser will show results as in Figure 2 below. The value of target variables should be as follows:
      i) **SatelliteSystem.Power_MOS=0.15**
      ii) **SatelliteSystem.Weight_MOS=0.05**



Name	Qualified Name	Type	Causality	Values
SatelliteSystem	Satellite::Instance01::satelliteS...	SatelliteSystem		
mass		Kilogram	ancillary	95
mass_MOS		Real	target	0.05
power_MOS		Real	target	0.15
control	Satellite::Instance01::satelliteS...	Control		
mCon		Kilogram	given	5
pCon		Kilowatt	given	1.5
instrumentation	Satellite::Instance01::satelliteS...	Instrumentation		
mIns		Kilogram	given	20
pIns		Kilowatt	given	2
powerSystem	Satellite::Instance01::satelliteS...	PowerSystem		
mPow		Kilogram	given	20
pPow		Kilowatt	ancillary	8.5
propulsion	Satellite::Instance01::satelliteS...	Propulsion		
mPro		Kilogram	given	50
pPro		Kilowatt	given	5

Name	Local	Redef...	Relation	Active
m_mos1	Y		mass_MOS=(100-mass)/100	✓
mbal1	Y		mass=propulsion.mPro+instrumentation.mIns+control.mCon+powerSy...	✓
p_mos1	Y		power_MOS=(10-powerSystem.pPow)/10	✓
pdem1	Y		powerSystem.pPow=propulsion.pPro+instrumentation.pIns+control.p...	✓

Figure 2: ParaMagic® browser showing Satellite model instance (solved state)

3.2 Second Pass – create new models

It is useful to identify several types of users who work with SysML parametrics:

- Type 1: Someone who works with an existing model, including executing it and performing additional instance-oriented interactions such as solving, modifying values, changing causalities, and re-solving.
  - This type of user needs the least amount of SysML and parametrics know-how.
  - This a good place to start for the casual user, for someone wanting to do basic demos, or someone just beginning to explore SysML parametrics.
- Type 2: Someone who modifies the structure of an existing model and/or creates new instances.
  - This type of user requires more know-how.

    - They also need a fair amount of MagicDraw SysML tool-aided modification support (in some respects more than Type 3 users).
    - This is a good step towards becoming a Type 3 user.
- Type 3: Someone who creates their own model structures and instances from scratch or from pre-existing building blocks from a library.
    - This type of user requires a fair amount of know-how and needs good MagicDraw SysML support.
- Type 4: Someone who creates building block libraries that Type 2 and Type 3 users can utilize.
    - This type of user requires similar skills as Type 3, but with a bent towards making their work reusable and modular, as well as providing good documentation and rigorous validation.

The First Pass in the previous section provided a quick introduction for Type 1 users. After completing the First Pass, you can work at the Type 1 level with all the pre-built tutorial models and examples provided in this release—see a listing of these models in Section 4 of this document.

After completing the First Pass, you also have a good "big picture" basis to now proceed with the step-by-step tutorials (see section 5.2). After completing the tutorials, you will have achieved a good foundation to work as a Type 3 user. There are other topics you may eventually need that may be addressed in future tutorials and/or courses.

4 INSTALLATION

4.1 Installation Requirements

4.1.1 System Requirements

1) Operating system: ParaMagic® has been tested to work[9] with MagicDraw SysML (i.e. MagicDraw UML + SysML plugin) installed on the following operating systems:
    a) Windows 7 64-bit edition
    b) Mac OS X 10.8.5 (Mountain Lion) 64-bit edition
    c) Linux - Ubuntu 10.4 LTS
2) Java: ParaMagic® requires Java 1.7 or higher[10]. ParaMagic® uses the same Java installation that is being used by MagicDraw. To check the Java version used by MagicDraw, check that the value of the JAVA_HOME variable in **<MD_Root>\bin\magicdraw.properties** file points to Java 1.7 (JRE or JDK 7) installation on your computer. For example, **JAVA_HOME=C\:\\Program Files\\Java\\jre6** (or **jre7**)
3) Hard disk space: ParaMagic® requires 40 MB of hard disk space for installation.
4) RAM: ParaMagic® requires 500 MB of memory. Additional available RAM will improve the performance of the plugin.

4.1.2 MagicDraw Requirements

1) ParaMagic® 18.0 requires MagicDraw SysML 17.0.5 (or higher).
2) ParaMagic® 18.0 has been tested to work with MagicDraw SysML 17.0.5 and 18.0.

[9] Unless otherwise specified, ParaMagic® may work with other editions of these OS but it is not been rigorously tested for editions other than those mentioned here.
[10] Note that ParaMagic® no longer supports Java 1.6. Oracle stopped releasing public updates for Java 1.6 in F 2013 - http://www.oracle.com/technetwork/java/eol-135779.html

4.1.3 Core Solver Requirements

ParaMagic® allows users to select Mathematica / PlayerPro, OpenModelica[11], or MATLAB SMT[12] as the core solver—see section 4.2 (Step 3) for installation and configuration details. ParaMagic® has been tested with Mathematica 9, PlayerPro 9, OpenModelica 1.9.0, and MATLAB/Simulink R2014a.

4.2 Installation Process

Before installing the ParaMagic® plugin, make sure that you have administrative privileges to install new software on your computer. Follow the steps below to install ParaMagic® plugin.

Step 1. Start MagicDraw (MD) as Administrator
If you are running MD, close it. Start MD from a user account with admin privileges on your computer. On Windows 7 you must run MD as an administrator to install/uninstall plugins even if you have admin privileges on your account. To do this, right click on the MD application and select **Run as administrator**.

Step 2. Install the ParaMagic® plugin

a) If you <u>do not</u> have the ParaMagic® plugin installation (zip file),
- i) On MD's main menu bar, select **Help > Resource/Plugin Manager**
- ii) Click on **Check for Updates** button
- iii) You will see ParaMagic® plugin version 18.0 available for installation. Select this plugin.
- iv) Click on **Download / Install** button to install the ParaMagic® plugin.
- v) After installation, restart MD for the plugin to take effect.

b) If you have the ParaMagic® plugin installation (zip file),
- i) In the main menu bar, select **Help > Resource/Plugin Manager**
- ii) Click on the **Import** button and specify the location of the plugin zip file.
- iii) After installation, you will need to restart MD for the plugin to take effect. It is suggested that you finish all the steps below before restarting MD.
- iv) After installation, restart MD for the plugin to take effect.

Step 3. Select the core solver for ParaMagic®
ParaMagic® allows users to select Mathematica / PlayerPro, OpenModelica[13], or MATLAB SMT as the core solver. If you plan to use:

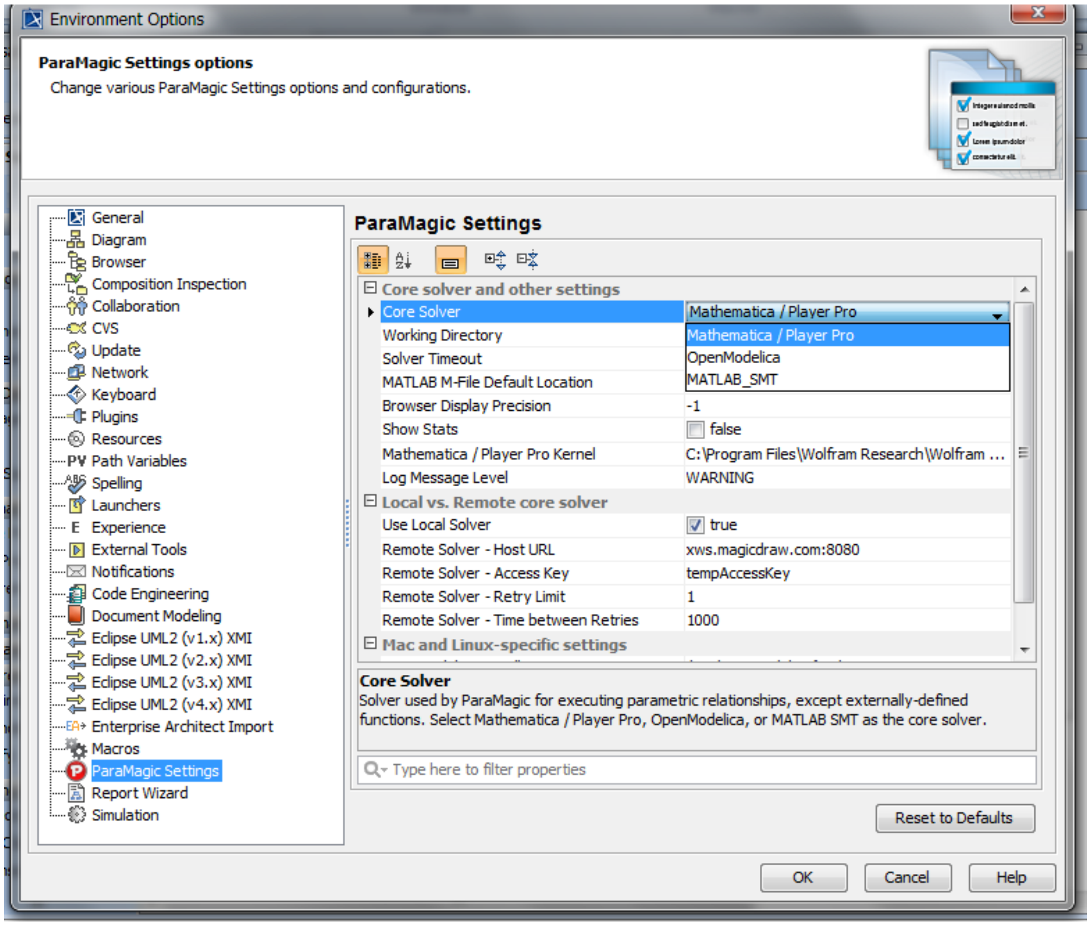
- Mathematica or PlayerPro as your core solver, follow steps in sub-section **a** below and then move to Step 4.
- OpenModelica as your core solver, follow steps in sub-section **b** below and then move to Step 4.
- MATLAB SMT as your core solver, follow steps in sub-section **c** below and then move to Step 4.

[11] OpenModelica: https://openmodelica.org/
[12] MATLAB Symbolic Math Toolbox: http://www.mathworks.com/products/symbolic/
[13] OpenModelica: https://openmodelica.org/

a) Using Mathematica or PlayerPro as your core solver

i) In MagicDraw, go to **Options > Environment** and select **ParaMagic Settings**, as shown in Figure 3.

ii) Set **Core Solver** variable to Mathematica / PlayerPro. If you do not see Mathematica / PlayerPro option for the Core Solver but only see Mathematica, click on the **Reset to Defaults** button at the bottom right hand corner of the Settings window.


Environment Options

ParaMagic Settings options
Change various ParaMagic Settings options and configurations.

General, Diagram, Browser, Composition Inspection, Collaboration, CVS, Update, Network, Keyboard, Plugins, Resources, Path Variables, Spelling, Launchers, Experience, External Tools, Notifications, Code Engineering, Document Modeling, Eclipse UML2 (v1.x) XMI, Eclipse UML2 (v2.x) XMI, Eclipse UML2 (v3.x) XMI, Eclipse UML2 (v4.x) XMI, Enterprise Architect Import, Macros, ParaMagic Settings, Report Wizard, Simulation

ParaMagic Settings

Core solver and other settings	
Core Solver	Mathematica / Player Pro (Mathematica / Player Pro, OpenModelica, MATLAB_SMT)
Working Directory	
Solver Timeout	
MATLAB M-File Default Location	
Browser Display Precision	-1
Show Stats	false
Mathematica / Player Pro Kernel	C:\Program Files\Wolfram Research\Wolfram ...
Log Message Level	WARNING
Local vs. Remote core solver	
Use Local Solver	✓ true
Remote Solver - Host URL	xws.magicdraw.com:8080
Remote Solver - Access Key	tempAccessKey
Remote Solver - Retry Limit	1
Remote Solver - Time between Retries	1000
Mac and Linux-specific settings	

Core Solver
Solver used by ParaMagic for executing parametric relationships, except externally-defined functions. Select Mathematica / Player Pro, OpenModelica, or MATLAB SMT as the core solver.

Figure 3: ParaMagic® Settings window

iii) For short-term (30 day) evaluation, you can use our test Mathematica server. For long-term production usage, you should use your own Mathematica license.

iv) To use our test Mathematica server for short-term evaluation, follow the steps below. Note that you will need an internet connection and your enterprise firewall may block connection to the Mathematica server.

(1) In the ParaMagic Settings window (Figure 3) locate the second section – *Local vs. Remote core solver.*
(2) Set **Use Local Solver** variable to **false**
(3) Check that the **Remote Solver – Host URL** variable is set to **xws.magicdraw.com:8080**
(4) Check that the **Remote Solver – Access Key** variable is set to **tempAccessKey**
(5) Click OK and close the ParaMagic Settings window.

v) If you are using your own Mathematica or PlayerPro license, follow the steps below.

(1) Check that Mathematica or PlayerPro is installed correctly by trying to launch it from your Start menu (Windows) or Finder (Mac).

(2) In MagicDraw, go to **Options > Environment** and select **ParaMagic Settings**, as shown in Figure 3.

(3) Under the **Core solver and other settings** section, locate the **Mathematica / PlayerPro Kernel** property.

(4) Click on the value field of the **Mathematica / PlayerPro Kernel** property and then select the file chooser button, as highlighted in Figure 4. Browse and select the **MathKernel** executable file.

(a) For Windows 7, this is typically **C:\Program Files\Wolfram Research\Wolfram Player Pro\9.0\MathKernel.exe**,

(b) For Mac OS X, this is typically **/Applications/Wolfram Player Pro.app.app/Contents/MacOS/MathKernel.**

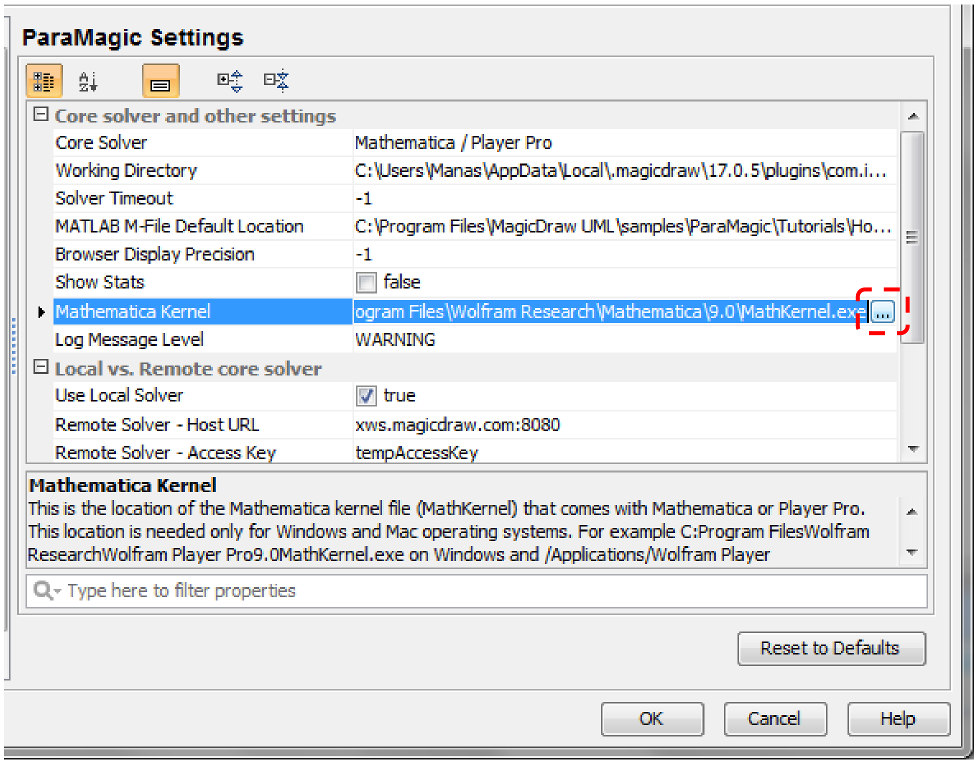
(c) For Linux, the **Mathematica / PlayerPro Kernel** property is not used, and hence does not need to be populated.

However, a symbolic link (shortcut) **/usr/local/bin/math** must exist and point to the math kernel that comes with Mathematica or PlayerPro. If you are using Mathematica on Linux, this link is typically created by the Mathematica installation process and must exist already. Verify the same. If you are using PlayerPro on Linux and don't have Mathematica installed, then this link will not exist and must be created using the command below.

```
sudo ln -s /usr/local/Wolfram/PlayerPro/9.0/Executables/math /usr/local/bin/math
```

The first path in the command (**/usr/local/Wolfram/PlayerPro/9.0/Executables/math**) is the location of the math kernel file in the PlayerPro installation. If you have PlayerPro installed in a different location, update this path in the command. The second path in the command (**/usr/local/bin/math**) is the full path of the softlink that will be created. Do not modify this. For more details on creating symbolic link, refer to http://manpages.ubuntu.com/manpages/utopic/man1/ln.1.html

(5) Under the **Local vs. Remote core solver** section, set **Use Local Solver** variable to **true**



ParaMagic Settings

Core solver and other settings

Core Solver	Mathematica / Player Pro
Working Directory	C:\Users\Manas\AppData\Local\.magicdraw\17.0.5\plugins\com.i...
Solver Timeout	-1
MATLAB M-File Default Location	C:\Program Files\MagicDraw UML\samples\ParaMagic\Tutorials\Ho...
Browser Display Precision	-1
Show Stats	false
Mathematica Kernel	ogram Files\Wolfram Research\Mathematica\9.0\MathKernel.exe
Log Message Level	WARNING

Local vs. Remote core solver

Use Local Solver	true
Remote Solver - Host URL	xws.magicdraw.com:8080
Remote Solver - Access Key	tempAccessKey

Mathematica Kernel
This is the location of the Mathematica kernel file (MathKernel) that comes with Mathematica or Player Pro. This location is needed only for Windows and Mac operating systems. For example C:Program FilesWolfram ResearchWolfram Player Pro9.0MathKernel.exe on Windows and /Applications/Wolfram Player

Figure 4: Click on the file chooser to specify the location of MathKernel executable file for Mathematica or PlayerPro installation

(6) Click OK in the **ParaMagic Settings** window.

Note: Before using Mathematica (or PlayerPro) with ParaMagic®, users must launch Mathematica (or PlayerPro) to verify that the product is activated and that the user has a valid license.

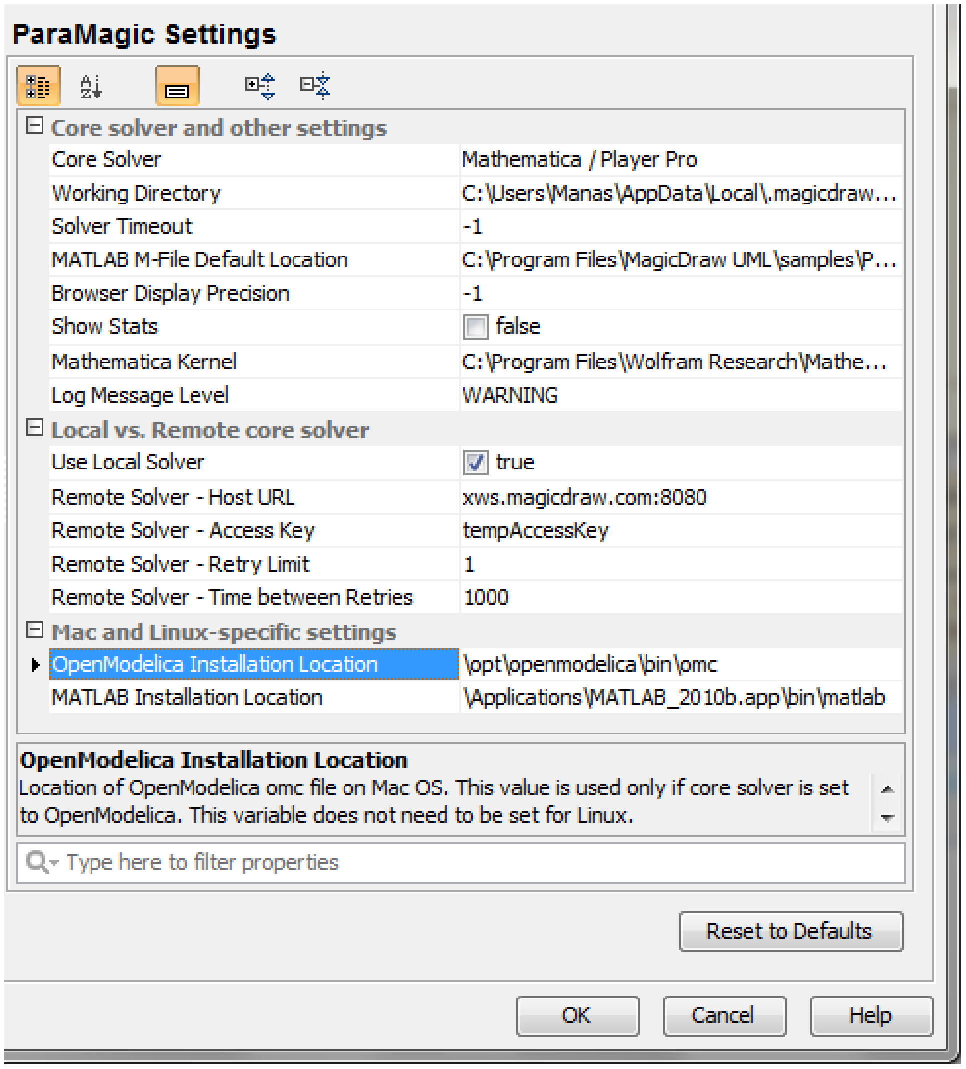
b) Using OpenModelica as your core solver

OpenModelica is a *free* Modelica-based modeling and simulation tool available as part of the OpenModelica Project that is managed by the Open Source Modelica Consortium (OSMC).

i) Download and install OpenModelica 1.9.0
   (1) **Windows** – Download and run the OpenModelica-1.9.0-revision-17628.exe file located here: https://build.openmodelica.org/omc/builds/windows/releases/1.9.0/
   (2) **Mac** – Download the openmodelica-devel-8719-tag-1.7.0.dmg file locate here: http://build.openmodelica.org/omc/builds/mac/binaries/. OpenModelica on Mac requires XCode. For details, see http://www.openmodelica.org/index.php/download/download-mac
   (3) **Linux** – Follow the installation instructions located here: https://openmodelica.org/index.php/download/download-linux

ii) In MagicDraw, go to **Options > Environment** and select **ParaMagic Settings**, as shown in Figure 5 below.
iii) Set **Core Solver** variable to OpenModelica.
iv) If you are on Mac OS, set the **OpenModelica Installation Location** variable (Figure 5) to the location of omc file in your OpenModelica installation, for e.g. **/opt/openmodelica/bin/omc**.

Copyright © 2014 InterCAX LLC. All Rights Reserved.



ParaMagic Settings

⊟ Core solver and other settings	
Core Solver	Mathematica / Player Pro
Working Directory	C:\Users\Manas\AppData\Local\.magicdraw...
Solver Timeout	-1
MATLAB M-File Default Location	C:\Program Files\MagicDraw UML\samples\P...
Browser Display Precision	-1
Show Stats	☐ false
Mathematica Kernel	C:\Program Files\Wolfram Research\Mathe...
Log Message Level	WARNING
⊟ Local vs. Remote core solver	
Use Local Solver	☑ true
Remote Solver - Host URL	xws.magicdraw.com:8080
Remote Solver - Access Key	tempAccessKey
Remote Solver - Retry Limit	1
Remote Solver - Time between Retries	1000
⊟ Mac and Linux-specific settings	
OpenModelica Installation Location	\opt\openmodelica\bin\omc
MATLAB Installation Location	\Applications\MATLAB_2010b.app\bin\matlab

OpenModelica Installation Location
Location of OpenModelica omc file on Mac OS. This value is used only if core solver is set to OpenModelica. This variable does not need to be set for Linux.

Figure 5: For Mac OS, set the variable OpenModelica Installation Location

v) If you are on Windows or Linux, you do not need to set anything else after installing OpenModelica in step iii above.
vi) Click OK in the **ParaMagic Settings** window.

c) Using MATLAB Symbolic Math Toolbox (SMT) as your core solver

i) Ensure that MATLAB and Symbolic Math Toolbox are installed on your machine.
ii) In MagicDraw, go to **Options > Environment** and select **ParaMagic Settings**, as shown in Figure 5 above.
iii) Set **Core Solver** variable to MATLAB_SMT.
iv) If you are on Mac or Linux, set the **MATLAB Installation Location** variable to the location of the MATLAB executable file, for e.g. **/Applications/MATLAB_R2014a.app/bin/matlab** (for Mac) or **/home/intercax/MATLAB/R2014a/bin/matlab** (for Linux)
v) If you are on Windows, you do not need to set anything else.
vi) Click OK in the **ParaMagic Settings** window.

Step 4. Close and Restart MagicDraw (MD) as an Administrator

After restarting MD, your installation is configured to use the ParaMagic® 18.0 plugin.

Copyright © 2014 InterCAX LLC. All Rights Reserved.

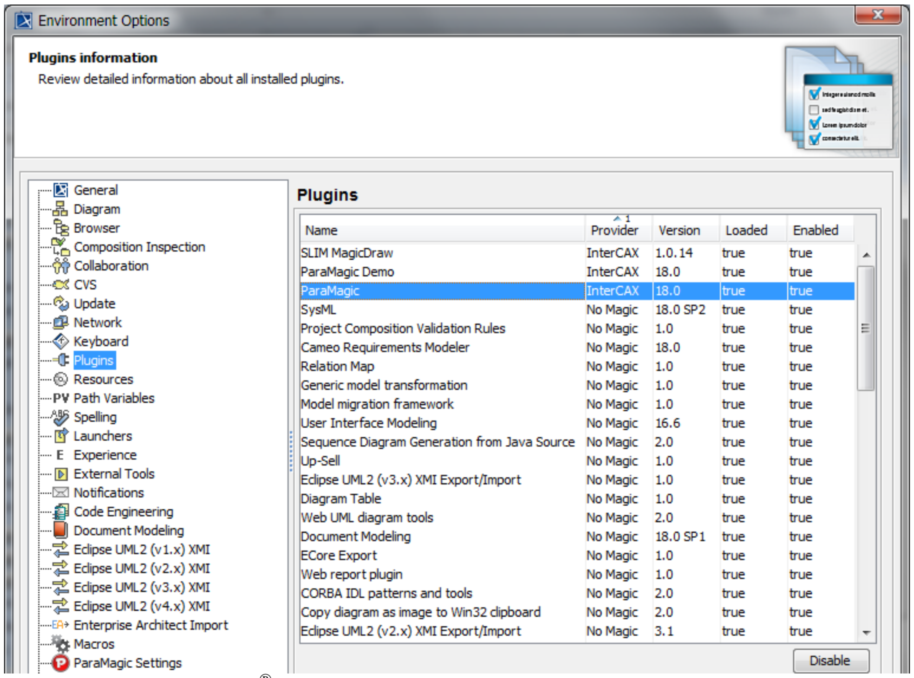
Step 5. Test ParaMagic® installation

a) In MagicDraw, select **Options > Environment**. Select **Plugins** on the LHS, as shown in Figure 6 below, and click on the **Provider** column to sort the list by providers. You should see ParaMagic® 18.0 in the list of installed plugins. Verify that the **Loaded** and **Enabled** properties are set to true for the ParaMagic® plugin.

b) Open and execute the Addition model and the Satellite model as shown in section 2 above.

i) During solution, ParaMagic® will indicate the core solver that is being used for solving. Verify that this is same as the core solver your setup and selected in Step 3 above.

ii) Ensure that you get the same solution results as shown in section 2 (Figure 1 and Figure 2).


Environment Options

Plugins information

Review detailed information about all installed plugins.

- General
- Diagram
- Browser
- Composition Inspection
- Collaboration
- CVS
- Update
- Network
- Keyboard
- Plugins
- Resources
- Path Variables
- Spelling
- Launchers
- Experience
- External Tools
- Notifications
- Code Engineering
- Document Modeling
- Eclipse UML2 (v1.x) XMI
- Eclipse UML2 (v2.x) XMI
- Eclipse UML2 (v3.x) XMI
- Eclipse UML2 (v4.x) XMI
- Enterprise Architect Import
- Macros
- ParaMagic Settings

Plugins

Name	Provider	Version	Loaded	Enabled
SLIM MagicDraw	InterCAX	1.0.14	true	true
ParaMagic Demo	InterCAX	18.0	true	true
ParaMagic	InterCAX	18.0	true	true
SysML	No Magic	18.0 SP2	true	true
Project Composition Validation Rules	No Magic	1.0	true	true
Cameo Requirements Modeler	No Magic	18.0	true	true
Relation Map	No Magic	1.0	true	true
Generic model transformation	No Magic	1.0	true	true
Model migration framework	No Magic	1.0	true	true
User Interface Modeling	No Magic	16.6	true	true
Sequence Diagram Generation from Java Source	No Magic	2.0	true	true
Up-Sell	No Magic	1.0	true	true
Eclipse UML2 (v3.x) XMI Export/Import	No Magic	1.0	true	true
Diagram Table	No Magic	1.0	true	true
Web UML diagram tools	No Magic	2.0	true	true
Document Modeling	No Magic	18.0 SP1	true	true
ECore Export	No Magic	1.0	true	true
Web report plugin	No Magic	1.0	true	true
CORBA IDL patterns and tools	No Magic	2.0	true	true
Copy diagram as image to Win32 clipboard	No Magic	2.0	true	true
Eclipse UML2 (v2.x) XMI Export/Import	No Magic	3.1	true	true

Figure 6: Verify that ParaMagic® is listed as a plugin and Loaded and Enabled properties are set true

Congratulations! You are now setup to use ParaMagic®.

5 User Documents and Models

5.1 Users Guide

1) This document is the User Guide for ParaMagic® plugin
2) This Users Guide is also located here after installation: **<MD_Root>\manual\ParaMagic\Users_Guide.pdf**

5.2 Tutorials

1) This installation comes with 8 tutorials. Each tutorial has step-by-step instructions to create a valid SysML model that can be solved using this plugin. All tutorials are described in one document that is located here: **<MD_Root>\samples\ParaMagic\Tutorials\Tutorials.pdf**
2) Each tutorial also has a pre-built SysML model with instances that are ready to explore, solve, and perform trade studies on instance values. These models are located here: **<MD_Root>\samples\ParaMagic\Tutorials**

Copyright © 2014 InterCAX LLC. All Rights Reserved.

a) **Addition** is a SysML model for learning how to create basic parametric equations. Its tutorial includes step-by-step instructions and demonstrates an addition operation applied to system properties.
b) **CommNetwork** is a SysML model of a communication network system.
c) **Electronics** is a SysML model of a network architecture.
d) **LittleEye** is a SysML model of a UAV-based road scanning system named LittleEye.
e) **LittleEyeTradeStudy** is a SysML model to demonstrate ParaMagic's trade study capabilities using the LittleEye system model (above).
f) **Satellite** is a SysML model of a satellite system.
g) **Orbital** is an early-stage orbital mechanics and spacecraft model in SysML that demonstrates the use of ParaMagic-Excel connection (section 8.1) and ParaMagic-Custom Mathematica connection (section 8.1).
h) **HomeHeating** is a SysML model of a home heating system that demonstrates the execution of Simulink models using the ParaMagic-MATLAB connection (section 8.3). This model has been tested to work with MATLAB 2014a.

5.3 Examples

1) This installation comes with 11 example SysML models that are ready to explore, solve, and perform trade studies on instance values.
2) These example SysML models are located here: **<MD_Root>\samples\ParaMagic\Other_Examples.** A document describing these example models is located here: **<MD_Root>\samples\ParaMagic\Other_Examples\Other_Examples.pdf**
3) The following example models are provided with ParaMagic™
   i) **Banking** is a SysML model of a banking services system.
   ii) **Circuit** is a SysML model of an electrical circuit and demonstrates the application of Ohm's Law.
   iii) **Financial** is a SysML model of financial projections for a small business.
   iv) **Linkage Systems** (formerly FlapLinkage) is a SysML model of mechanical linkage system—a part used in an airframe structure. It demonstrates the use of SysML for modeling design constraints as well as building block libraries and other modeling concepts. See item 0 below for a corresponding paper (Part 2) that overviews the theory behind SysML parametrics as well as applications to simulation templates.
   v) **Insurance** is a SysML model of a web-based insurance claim filing system.
   vi) **OpAmp** is a SysML model of an operational amplifier. By using the same constructs as in the **Circuit** model, it demonstrates the reusability of concepts defined in SysML.
   vii) **ProjectPlan** is a SysML model of a project scheduling system and demonstrates the use of conditionals (if-then statements).
   viii) **SpringSystems** is a SysML model of a system of two springs connected in series and exhibiting linear deformation behavior. See item 0 below for a corresponding paper (Part 2) that overviews the theory behind SysML parametrics as well as applications to simulation templates.
   ix) **Trade** is a SysML model of a trade financing system.
   x) **VehicleMassRollup** is a SysML model of a vehicle system and demonstrates how to model variant architectures using recursion and redefinition and execute the same using ParaMagic®.
   xi) **MATLAB_Example** is a SysML model illustrating the different ways to connect to MATLAB functions and scripts using ParaMagic®. See the package diagram *Scripts and Functions* for details.

Note that the intent of the tutorial and example models is to present how SysML (Parametrics in particular) can be used for different types of problems in different domains. Some models are created for demonstration purposes only, and not intended to represent all aspects of the systems in the most accurate manner.

5.4 Relevant Publications

1) Bajaj, M., Zwemer, D., Peak, R., Phung, A., Scott, A. and Wilson, M. (2011). *Satellites to Supply Chains, Energy to Finance — SLIM for Model-Based Systems Engineering, Part 1: Motivation and Concept of SLIM*. 21st Annual INCOSE International Symposium, Denver, CO, June 20-23, 2011. (available at http://www.omgsysml.org/SLIM_for_MBSE_Bajaj_Part1.pdf)

2) Bajaj, M., Zwemer, D., Peak, R., Phung, A., Scott, A. and Wilson, M. (2011). *Satellites to Supply Chains, Energy to Finance — SLIM for Model-Based Systems Engineering, Part 2: Applications of SLIM*. 21st Annual INCOSE International Symposium, Denver, CO, June 20-23, 2011. (available at http://www.omgsysml.org/SLIM_for_MBSE_Bajaj_Part2.pdf)

3) Peak, R.S., Roger M. Burkhart, Sanford A. Friedenthal, Wilson, M.W., Bajaj, M. and Kim, I. (2007). *Simulation-Based Design Using SysML Part 1: A Parametrics Primer*. The Seventeenth International Symposium of the International Council on Systems Engineering, San Diego, California, USA June 24 -28, 2007. (available at http://eislab.gatech.edu/pubs/conferences/2007-incose-is-1-peak-primer/)

4) Peak, R.S., Burkhart, R.M., Friedenthal, S.A., Wilson, M.W., Bajaj, M. and Kim, I. (2007). *Simulation-Based Design Using SysML Part 2: Celebrating Diversity by Example*. The Seventeenth International Symposium of the International Council on Systems Engineering, San Diego, California, USA June 24 -28, 2007. (available at http://eislab.gatech.edu/pubs/conferences/2007-incose-is-2-peak-diversity/)

6 SysML Model Requirements

This section consists of a list of modeling requirements that need to be satisfied to use the solver capabilities of ParaMagic® plugin. Some of these requirements are based on recommended practices for enhanced model interoperability; some are to make SysML models less ambiguous for the plugin and Mathematica / PlayerPro / OpenModelica / MATLAB SMT solvers; while others are limitations of this version of the plugin and/or supported solvers. These guidelines are followed in the tutorial examples included with this installation. It is suggested that a user walks through the tutorials first and then review these requirements for better understanding.

6.1 Structural requirements

These requirements deal with the model schema and instances created using SysML. In our terminology, a *schema* defines the structure of the model using SysML constructs such as blocks, properties, and constraint blocks; and an *instance* conforms to this structure and has populated slots (properties)—completely or partially. There may be several instance models for a given schema. A system model *schema* represents a family of system alternatives (topological or parametric variations) while a system model *instance* represents a specific system alternative that is analyzed and evaluated.

6.1.1 Model schema requirements

1) ParaMagic® requires that all SysML elements required for defining the schema must be in packages.
2) If a constraint property (say `c1`) defined in a block has the same name as an inherited constraint property, ParaMagic® will override the inherited constraint property with the constraint property `c1`. Modelers can use this feature to define abstract constraint properties for a block that are implemented or overridden by constraint properties of the block's subtypes.

6.1.2 Model instance requirements

1) Instance slots that correspond to block properties participating in parametric relations must be initialized/populated unless the lower bound of the multiplicity for these properties is equal to 0 (zero). ParaMagic® provides capabilities to automatically initialize slots.
2) Instance slots corresponding to value properties can be populated by LiteralReal, LiteralInteger, and LiteralString value specifications.
3) All numeric values in instance slots must begin with a number (0-9). Values beginning with a space or decimal point may not be read correctly.
4) If the value property corresponding to a slot is connected to the outputs of multiple ONEWAY relations, then the slot must have causality "given". This restriction is imposed to prevent instance models from being over-constrained.
5) *Causality Verification and Assignment*: The truth table below states valid causality assignments for slots in a SysML instance model. Causality can be set and is validated for only those slots whose corresponding value properties participate in parametric relations. The validity of a causality assignment is based on whether or not a slot has value(s). In the table, TRUE implies valid assignments and FALSE implies invalid assignments. For example, if a slot has values and its corresponding value property participates in parametric relations, then its causality may be **given**, **ancillary**, or **target** but not **undefined**. Its causality may be **ancillary** or **target** only when the model is in a solved state. Similarly, if a slot has no values (empty), then its causality may be **undefined** or **target** but not **given** or **ancillary**.

Table 1: Truth table for valid causality assignments

	causality			
If a slot	**given**	**undefined**	**ancillary**	**target**
has values	TRUE	FALSE	TRUE	TRUE
does not have values (empty)[14]	FALSE	TRUE	FALSE	TRUE

ParaMagic® checks for validity of causality assignments when users attempt to browse SysML instance models in the ParaMagic® browser. The causality assignment utility in ParaMagic® that is invoked on instance models (**ParaMagic > Util > Add default causalities**) assigns default causalities based on this table.

6) Slots corresponding to value properties that are typed by Integer (or its subtype) should have causality "given", i.e. they can only be inputs to parametric calculations. ParaMagic® treats integers (only allowed as inputs) as real numbers during solving.

6.2 Naming requirements

1) All SysML blocks, constraint blocks, and properties (part / reference / shared / value) should have a name.
2) Block properties and constraint parameters should not have names that are reserved math keywords. In addition to names of math constants and functions listed in section 5.4, the following names are also reserved: **binom, str.**
3) All SysML model elements should have names that start with an alphabetical character (A-Z, a-z).

[14] "Empty" denotes that the slot (or slot value) has been activated but does not contain any value (i.e. its value is an empty string and represented as a Literal String). Note that if the slot (or slot value) is not activated, then causality cannot be assigned to it anyway.

4) The allowable character classes for naming model elements are: A-Z, a-z, 0-9, underscore, and whitespace.
5) Period (.) should not be used in naming model elements. Some of these limitations are due to math parsers and solvers.

6.3 Mathematical expression requirements

1) Mathematical functions (e.g., min, exp) should begin with a lower-case letter and use standard parentheses ( ) to enclose their arguments.
2) A valid mathematical equation for ParaMagic® processing purposes is one that has a single variable on the LHS. For example, **a=b+c** will be processed but users will face issues processing the same equation defined as **b+c=a**.
3) The list of math operators supported[15] in ParaMagic® are as follows:
    a) Addition (+), Subtraction (-), Multiplication (*), Division (/)
    b) Unary plus (+x), and Unary Minus (-x)
4) It is recommended that large expressions be broken into simpler expressions that are more readily solvable by Mathematica / PlayerPro / OpenModelica / MATLAB SMT. For example, the expression below

    ```
    k = (0.577*3.14*E*d) / ln(((1.15*t+D-d)*(D+d)) / ((1.15*t+D-d)*(D-d)))
    ```

    can be broken into two expressions:

    ```
    k = (0.577*3.14*E*d) / ln(dummyVariable)
    dummyVariable=((1.15*t+D-d)*(D+d)) / ((1.15*t+D-d)*(D-d))
    ```

    Since ParaMagic® supports only one constraint expression per constraint block, this would require you to create two constraint blocks (one for each expression) and to create a dummy value property (corresponding to the **dummyVariable**) in the context block.

In general, the math expression syntax supported by ParaMagic® is based on JEP[16] (with extensions).

6.4 Value types and QUDV profile

ParaMagic® requires that value properties and constraint parameters participating in parametric constraint relations satisfy ONE OF the following requirements. Let the value type of the given value property (or constraint parameter) be V. Then,
1. V should be the value type “Real” (**SysML Profile::Blocks::Real**) or its subtype, OR
2. V should be the value type “Integer” (**SysML Profile::Blocks::Integer)** or its subtype, OR
3. V should be a value type with the quantityKind field populated. Such a value type represents a “quantity” which per SysML 1.3 can be measured (is quantifiable) using a number. See SysML 1.3 specification, section 8.3.2.10, page 50.

Requirement 3 above allows ParaMagic® users to leverage the standard value types now available with the SI ValueType library in MagicDraw (**QUDV Library::SI ValueType Library**).

Note that value properties typed by String (**SysML Profile::Blocks::String**) are handled by ParaMagic® but cannot participate in parametric relations.

[15] Note that ParaMagic™ may not warn if other operators are used. Users should only use the math operators stated here.
[16] JEP: http://www.singularsys.com/jep/

ParaMagic® has the following limitations regarding value types:

1. Internally, ParaMagic® treats all value types obeying the requirements above as real numbers.
2. ParaMagic® does not check for the compatibility of value types (or constraint parameters) connected using binding connectors. This verification is performed by MagicDraw. Hence, users should check and resolve warnings and errors generated by MagicDraw when connecting value properties and/or constraint parameters before launching ParaMagic®.
3. ParaMagic® does not perform automatic unit conversions.

The ParaMagic® browser only displays slots whose corresponding value properties are typed by Real, Integer, String, or a value type representing a quantity, and their subtypes.

6.5 Math constants and functions

ParaMagic® currently supports the following math constants and functions in defining constraint specifications of constraint blocks.

Some functions are not supported or have limited support if OpenModelica (OM) or MATLAB SMT is selected as the core solver. These functions have comments in blue (for OpenModelica) and red (for MATLAB SMT).

A list of constants is provided below.

Name	**Syntax**	**Example**
Pi(**π**)	**pi**	y = pi + x

A list of functions is provided below. In principle, several of these functions may be executed in different causalities (directions)—computing LHS value(s) for a given set of RHS values, or computing RHS value(s) for a given set of RHS and LHS values. In the current version of this plugin, these functions and expressions containing these functions are verified to work reliably (i.e. give a correct answer) only in the natural causality (computing LHS value from a given set of RHS values). Hence, these functions are marked as ONEWAY by default when used in a MagicDraw model. If you would like to try solving a function (or the expression it is being used in) in the reverse direction, uncheck the ONEWAY mark for that function (or expression) in the Relationship Browser section of the ParaMagic browser (see Figure 33 in section 7.2) and press Solve. Note that this selection will not be stored in ParaMagic® settings. This implies that when the browser is launched the next time, the ONEWAY marks for these functions need to be unchecked again. The two primary reasons for marking these functions as ONEWAY by default are:

2) For large expressions involving several functions, Mathematica may not always return an answer when these expressions are evaluated in non-natural directions.
3) The current version of ParaMagic® does not support inequalities. Without an inequality constraint, some of the functions (esp. trigonometric functions) return a general solution instead of a specific solution. For example **x=sin(pi/2)** will return the general solution **x = 2*n*pi + pi/2**. This limitation will be addressed in future release(s) of ParaMagic®.

If OpenModelica is selected as the core solver, none of the functions are marked as ONEWAY. However, this may result in an unexpected (but correct) answer if some of these functions are solved in different directions (due to lack of support for inequalities). For example, solving 1=Sin(x) may not always return x=1.570 (or pi/2).

Name	**Syntax**	**Example**
Sine	**sin(x)**	y = sin(x)
Cosine	**cos(x)**	y = cos(x)
Tangent	**tan(x)**	y = tan(x)

Copyright © 2014 InterCAX LLC. All Rights Reserved.

Arc Sine	**asin(x)**	y = asin(x)
Arc Cosine	**acos(x)**	y = acos(x)
Arc Tangent	**atan(x)**	y = atan(x)
Arc Tangent (gives the arc tangent of y/x, taking into account which quadrant the point (x, y) is in)	**atan2(x,y)**	z = atan2(x,y)
Hyperbolic Sine	**sinh(x)**	y = sinh(x)
Hyperbolic Cosine	**cosh(x)**	y = cosh(x)
Hyperbolic Tangent	**tanh(x)**	y = tanh(x)
Inverse Hyperbolic Sine Not supported for OM	**asinh(x)**	y = asinh(x)
Inverse Hyperbolic Cosine Not supported for OM	**acosh(x)**	y = acosh(x)
Inverse Hyperbolic Tangent Not supported for OM	**atanh(x)**	y = atanh(x)
Natural Logarithm (ln(x)) (where x is a positive real number)	**ln(x)**	y = ln(x)
Logarithm ($\log_b x$) (where b and x are positive real numbers) For OM, only log (10,x) is supported	**log(b,x)**	y = log(b,x)
Exponential	**exp(x)**	y = exp(x)
Absolute Value	**abs(x)**	y = abs(x)
Random number (between 0 and 1)	**rand()**	y = rand()
Modulus	**mod(x,y)**	z = mod(x,y)
Square Root	**sqrt(x)**	y = sqrt(x)
Power x^y	**pow(x,y)**	z = pow(x,y)
Round (rounds argument to the closest integer, or the closest even integer for arguments equidistant from two integers)	**round(x)**	y = round(x)
Ceil (rounds argument to the smallest integer greater than or equal to the argument)	**ceil(x)**	y = ceil(x)
Floor (rounds argument to the greatest integer less than or equal to the argument)	**floor(x)**	y = floor(x)
Min (returns the argument with minimum value)	**min(x_1,x_2,x_3,…)** For Mathematica, x_1,x_2,…can be arrays or single-valued For OM, x_1,x_2,…can only be single-valued For MATLAB SMT, use min(X) if X is an array, or min(x1,x2,x3,…) if x1,x2, x3,… are single-valued. Also see note[17] below.	y = min(x_1,x_2,x_3)
Max (returns the argument with maximum value)	**max(x_1,x_2,x_3,…)** For Mathematica, x_1,x_2,…can be arrays or single-valued For OM, x_1,x_2,…can only be single-valued For MATLAB SMT, use max(X) if X is an array or max(x1,x2,x3,…) if x1,x2, x3,…are single-valued. Also see note[17] below.	y = max(x_1,x_2,x_3)
Average (returns the arithmetic mean of	**avg(x)**	y = avg(x)

[17] For MATLAB SMT, the inputs to *min* and *max* functions should be givens before solving. The functions do not work if their inputs are computed by solving other equations simultaneously.

Copyright © 2014 InterCAX LLC. All Rights Reserved.

members of the array passed as argument)	where **x** is an array	
Sum (returns the sum of the members of the array passed as argument)	**sum(x)** where **x** is an array	y = sum(x)

6.6 Conditional Functions and Operators

ParaMagic® currently supports conditional statements in the following format.

<Result> = if(<Condition>, <Result if Condition = true>, <Result if Condition = false>)

For example, in the conditional statement
X2 = if(X1 > 0, X1, -X1)

X2 is set equal to **X1** when **X1** is positive and **–X1** when **X1** is negative. Hence, this condition is the equivalent of **X2 = abs(X1).**

The following operators can be used as part of the condition term

Operator	Syntax	Example
Equal to	==	y = if(x == 1, 2, 1)
Not Equal to	!=	y = if(x != 1, 2, 1)
Greater than	>	y = if(x > 1, 2, 1)
Less than	<	y = if(x < 1, 2, 1)
Greater than or Equal to	>=	y = if(x >= 1, 2, 1)
Less than or Equal to	<=	y = if(x <= 1, 2, 1)
AND	&&	y = if(x1 == 0 && x2 < 5, 2, 1)
OR	||	y = if(x1 == 0 || x2 < 5, 2, 1)
NOT	!	y = if(!(x>1), 2, 1)

For MATLAB SMT, conditional expressions do not work reliably when combined with other expressions. For e.g. **a = b + if(d>e, 1, 0)** will not work with MATLAB SMT. In these cases, users are requested to use multiple simpler expressions instead, for e.g. **a = b + c** and **c = if(d>e,1,0)**

6.7 Aggregate Properties and Functions

ParaMagic® also supports aggregate value properties of type **Real**, that is, properties that contain a list of one or more values of type **Real**.

6.7.1 Multiplicity

The default multiplicity for value properties is 1. In order to hold more than one value, the multiplicity of the value property must be set to 1..* , 0..*, or * in the model schema.

6.7.2 Instance Slot Values

When setting up an instance for solution, values are assigned to slots for each variable—either real values for **given** values or empty values as placeholders for unknowns. To add multiple values to a slot, create the first value as normal in the Instance Specification window, then use the Plus icon at the bottom left of the window to add additional values one at a time. Multiple empty slot values can be created in the same way.

6.7.3 Aggregate Functions and Operators

The following function and operators are supported by ParaMagic® for aggregate values.

Some functions are not supported or have a limited support if OpenModelica (OM) is selected as the core solver. These functions have comments in blue.

Function	**Explanation**
y = a[2]	a is an aggregate of Real numbers and y is a single Real number; y = second item in aggregate a; parameterized indices not supported (**a[i]** not supported)
y = sum(a)	a is an aggregate of Real numbers and y is a single Real number; y = a1 + a2 + a3 +…
y = avg(a)	a is an aggregate of Real numbers and y is a single Real number; y = mean of a1, a2, a3,…
y = standarddeviation(a) Not supported for OM	a is an aggregate of Real numbers and y is a single Real number; y = standard deviation of a1, a2, a3,…
y = variance(a) Not supported for OM	a is an aggregate of Real numbers and y is a single Real number; y = variance of a1, a2, a3,…
y = min(a)	a is an aggregate of Real numbers and y is a single Real number; y = minimum of a1, a2, a3,…
y = max(a)	a is an aggregate of Real numbers and y is a single Real number y = maximum of a1, a2, a3,…
x = a	x and a are aggregates of Real numbers; {x1, x2, x3,…} = {a1, a2, a3 …}. Note that the sizes of x and a must be same.
x = a + b	x, a, and b are aggregates of Real numbers; {x1, x2, x3…} = {a1+b1, a2+b2, a3+b3 …}; similar syntax for other math operators and functions. Note that the sizes of x, a, and b must be same.
x = sin(a)	x and a are aggregates of Real numbers; {x1, x2, x3…} = {sin(a1), sin(a2), sin(a3) …}; similar syntax for other trigonometric, exponential, logarithmic, and hyperbolic functions. Note that the sizes of x and a must be same.
x = n Not supported for OM	x is an aggregate of Real numbers and n is a single Real number; {x1, x2, x3…} = {n,n,n…}.
x = a + n Not supported for OM	x and a are aggregates of Real numbers, and n is a single Real number; {x1, x2, x3…} = {a1+n, a2+n, a3+n…}; similar syntax for other math operators and functions. Note that the sizes of x and a must be same.
x = pow(n,a) Not supported for OM	x and a are aggregates of Real numbers, and n is a single Real number; {x1, x2, x3…} = {n^a1,n ^a2,n^ a3 …}. Note that the sizes of x and a must be same.

For OpenModelica as the core solver, math expressions combining aggregates and single-valued variables are not supported. For example, **x = a + n**, where **x** and **a** are aggregates and n is a single-valued variable is not supported.

6.8 Complex Aggregate Relationships

ParaMagic® supports complex aggregate relationships. First, a short description of complex aggregates and complex aggregate relationships is presented to better characterize this feature of ParaMagic®.

A *primitive aggregate* is a collection of primitive elements, such as real numbers, strings, integers, etc. For example, in the relation **a = avg(b)**, where **a** is a single-valued real number and **b** is an array of real numbers, **b** is a primitive aggregate. Primitive aggregates are manifested in the SysML model when value properties have multiplicity > 1. ParaMagic® currently supports several types of relations involving primitive aggregates—see section 6.7 above.

A *complex aggregate* is a collection of blocks (in SysML context). Complex aggregates are manifested in a SysML model when part/reference/shared properties of a block have multiplicity > 1. For example, Figure 7 below illustrates a car system composed of several sub-systems, including the chassis sub-system. The Chassis is composed of several components. The parametric diagram in Figure 8 illustrates the relationship between the mass of the Chassis block and the masses of its components.

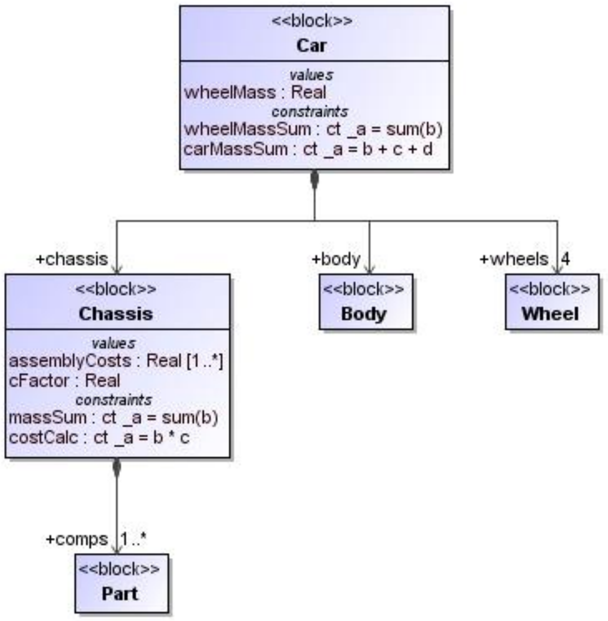


Figure 7: Car example to illustrate complex aggregates

The constraint relationship embodied in the **massSum** constraint property is an example of a complex aggregate relationship. This is because one of the parameters in the relationship is tied to a value property that is owned directly/indirectly by a complex aggregate (e.g. parameter **b** is tied to value property **mass** owned by complex aggregate **comps**).

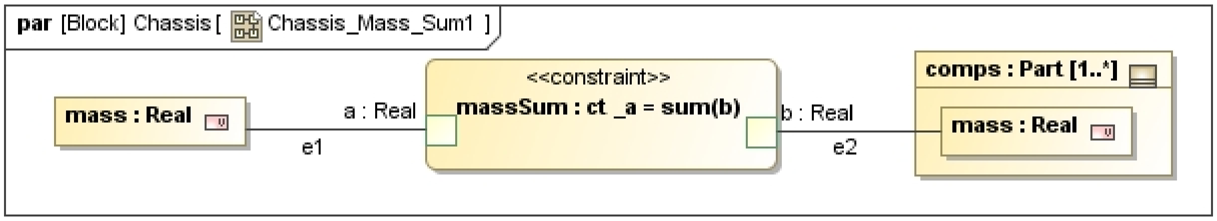


Figure 8: Parametric diagram illustrating the relationships between properties of the Chassis block (Figure 7)

In ParaMagic®, the depth of a complex aggregate relationship is defined as the maximum level of nesting of the properties (owned by complex aggregates) constrained by that relationship. For example, the depth of the complex aggregate relationship shown in Figure 8 above is equal to 1. The properties involved in the relationship are **mass** and **comps[i].mass**. The level of nesting of **mass** is 0 while that of **comps[i].mass** is 1. The level of nesting also corresponds to the number of complex aggregate part boundaries crossed by the binding connector that ties the parameters of the relationship with the block properties. For example, the connector **e2** in Figure 8 crosses only 1 complex aggregate boundary (of **comps**). Similarly, Figure 9 below illustrates a complex aggregate relationship (**loadSum**) with depth = 2. As evident, the connector **e2** crosses two complex aggregate part boundaries (**floors** and **parkedCars**). In Figure 9 below, if the multiplicity of **floors** was changed to 1, then the connector **e2** would cross only 1 complex aggregate part boundary (**parkedCars**). Hence, the depth of the complex aggregate relationship would change from 2 to 1.

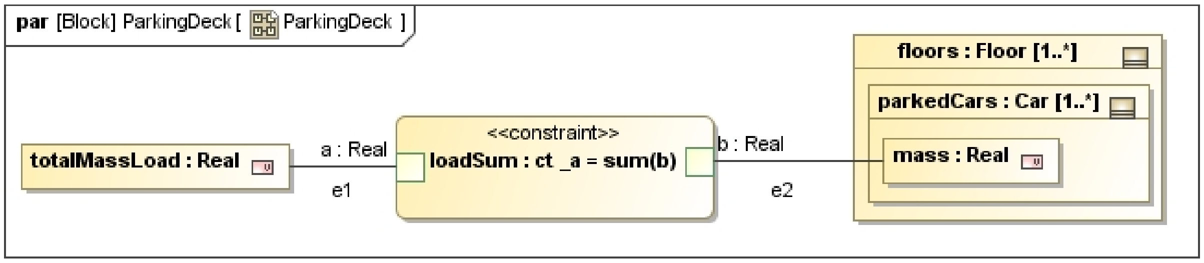


Figure 9: Parametric diagram illustrating a complex aggregate relationship of depth = 2.

ParaMagic® only supports execution of complex aggregate relationships with depth = 1. In general, the parameters of the complex aggregate relationship (and the properties they are connected to) can be single-valued or multi-valued (e.g. array). However, properties owned by complex aggregates and the relationship parameters they are tied to should be single-valued. For example, in the complex aggregate relationship illustrated in Figure 8, the property **comps[i].mass** is owned by the complex aggregate **comps**. Hence, **comps[i].mass** and the parameter **b** to which it is connected should be single-valued (as shown) for ParaMagic® to solve this relationship. Figure 10 below illustrates a complex aggregate relationship (**costCalc**) with depth = 1. ParaMagic® can solve this relationship because the property **comps[i].mass** owned by the complex aggregate **comps** is single-valued.

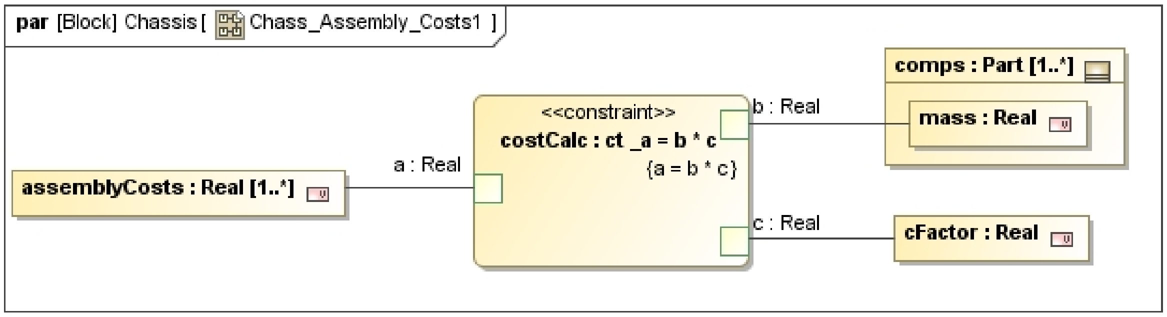


Figure 10: Parametric diagram illustrating a complex aggregate relationship with depth = 1.

Complex aggregate relationships can also involve constraint blocks that wrap MATLAB functions/scripts or Mathematica functions, as shown in Figure 11 below.

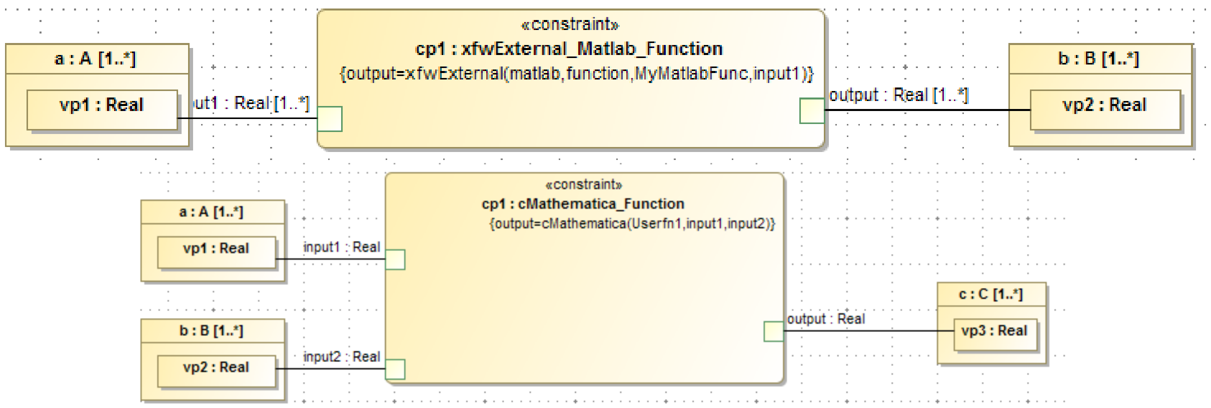


Figure 11: Complex aggregate relationships involving constraint blocks that wrap MATLAB function (top) and Mathematica function (bottom)

In some models, complex aggregate relationships involve primitive aggregates in the scope of complex aggregates. For example the parametric diagram below shows primitive aggregate *A1*, in the scope of complex aggregate *input*, participating in four complex aggregate relationships. In such cases, it is useful to have aggregate functions (e.g. **min**, **max**, **avg**, and **sum**) that can operate over primitive and complex aggregates. ParaMagic® provides this capability as shown in the table below. The model below shows how the new functions (**minca**, **maxca**, **avgca**, and **sumca**) are used.

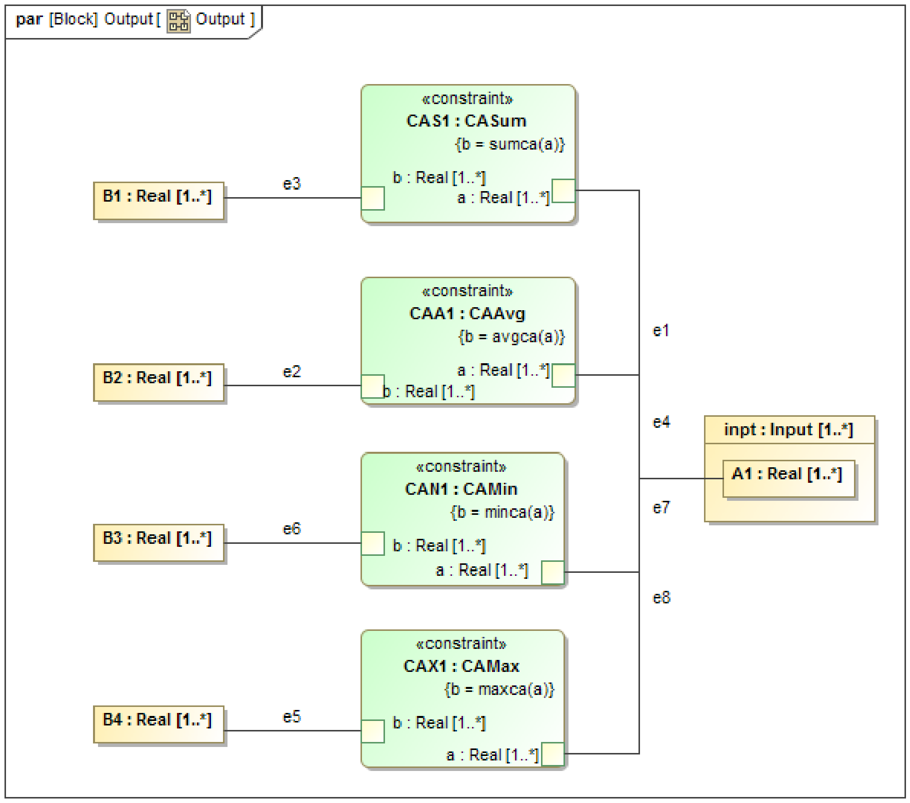


Figure 12: Special aggregate functions (sumca, avgca, minca, maxca) for complex aggregates

Function	Explanation
y = sum(input.x)	*y* and *x* primitive aggregates, and *input* is a complex aggregate This is mathematically equivalent to**:** **$y[i] = \Sigma_j$ (input[i].x[j])**, i.e. summation is performed over the primitive aggregate *x*.
y = sumca(input.x)	*y* and *x* primitive aggregates, and *input* is a complex aggregate This is mathematically equivalent to: **$y[j] = \Sigma_i$ (input[i].x[j])**, i.e. summation is performed over the complex aggregate *input*.
y = avg(input.x)	*y* and *x* primitive aggregates, and *input* is a complex aggregate This is mathematically equivalent to: **$y[i] = avg_j$ (input[i].x[j])**, i.e. average is performed over the primitive aggregate *x*.
y = avgca(input.x)	*y* and *x* primitive aggregates, and *input* is a complex aggregate This is mathematically equivalent to: **$y[j] = avg_i$ (input[i].x[j])**, i.e. average is performed over the complex aggregate *input*.
y = max(input.x)	*y* and *x* primitive aggregates, and *input* is a complex aggregate This is mathematically equivalent to: **$y[i] = max_j$ (input[i].x[j])**, i.e. max is performed over the primitive aggregate *x*.
y = maxca(input.x)	*y* and *x* primitive aggregates, and *input* is a complex aggregate This is mathematically equivalent to: **$y[j] = max_i$ (input[i].x[j]),** i.e. max is performed over the complex aggregate *input*.
y = min(input.x)	*y* and *x* primitive aggregates, and *input* is a complex aggregate This is mathematically equivalent to: **$y[i] = min_j$ (input[i].x[j]),** i.e. min is performed over the primitive aggregate *x*.
y = minca(input.x)	*y* and *x* primitive aggregates, and *input* is a complex aggregate This is mathematically equivalent to: **$y[j] = min_i$ (input[i].x[j])**, i.e. min is performed over the complex aggregate *input*.

6.9 Recursion and Redefinition

Recursion and redefinition are powerful concepts for modeling variant system architectures. With ParaMagic®, users can execute SysML parametric models involving redefinition and recursion to compute and compare measures-of-effectiveness (and/or key performance parameters) for variant architectures.

The **VehicleMassRollup** model included in ParaMagic® intallation (under **<MD Root>\samples\ParaMagic\Other_Examples**) is a SysML model of a vehicle system and demonstrates how to model variant architectures using recursion and redefinition and execute the same using ParaMagic®.

6.9.1 Recursion

Recursive Parametric Relationships are parametric constraints involving value properties of a block and value properties of its part/reference/shared properties that are of the same type as the block itself. Figure 13 illustrates an example recursive parametric relationship. The *Component* block has a part property (*sub components*) that is typed by the block itself—a component can have 0 or more sub components. The *Component* block has a value property (*mass*) and a constraint property (*mass rollup*). As shown in the parametric diagram in Figure 13, the *mass rollup* constraint relates the mass of a component with the masses of its sub components—mass of a component is the sum of the masses of its sub-components.

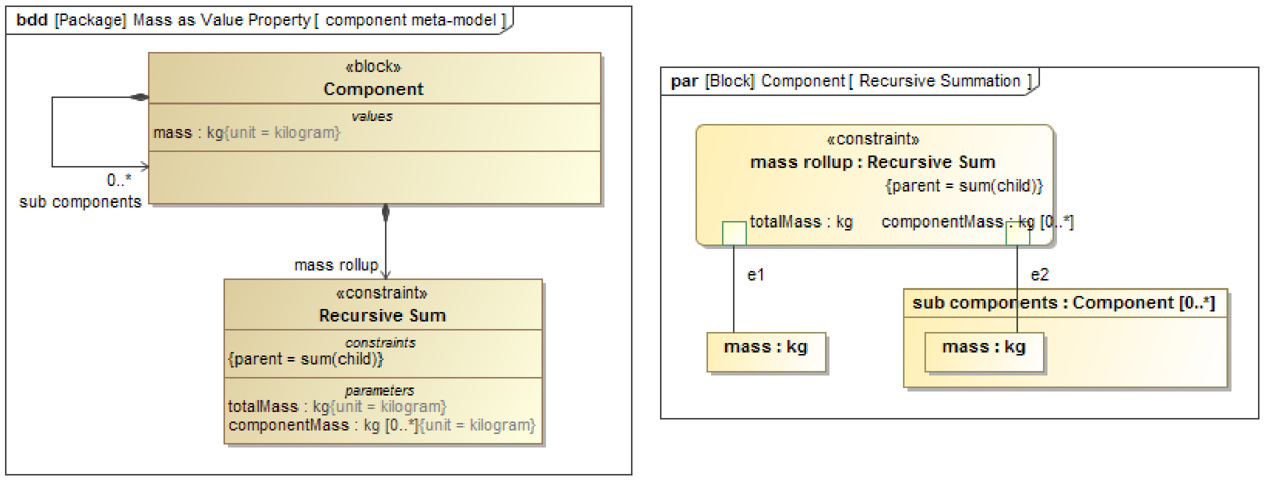

bdd [Package] Mass as Value Property [component meta-model]
«block»
Component
values
mass : kg{unit = kilogram}
0..*
sub components
mass rollup
«constraint»
Recursive Sum
constraints
{parent = sum(child)}
parameters
totalMass : kg{unit = kilogram}
componentMass : kg [0..*]{unit = kilogram}
par [Block] Component [Recursive Summation]
«constraint»
mass rollup : Recursive Sum
{parent = sum(child)}
totalMass : kg
componentMass : kg [0..*]
e1
e2
mass : kg
sub components : Component [0..*]
mass : kg

Figure 13: SysML block definition and parametric diagrams illustrating a recursive parametric relationship

Once a recursive parametric relationship has been defined at the block level, they can be executed by ParaMagic® in a recursive manner at the instance level. Figure 14 illustrates an instance structure conforming to the block structure shown in Figure 13. The *System* (instance of *Component*) is composed of 3 subsystems (instances of *Component*) which are then composed of other components.

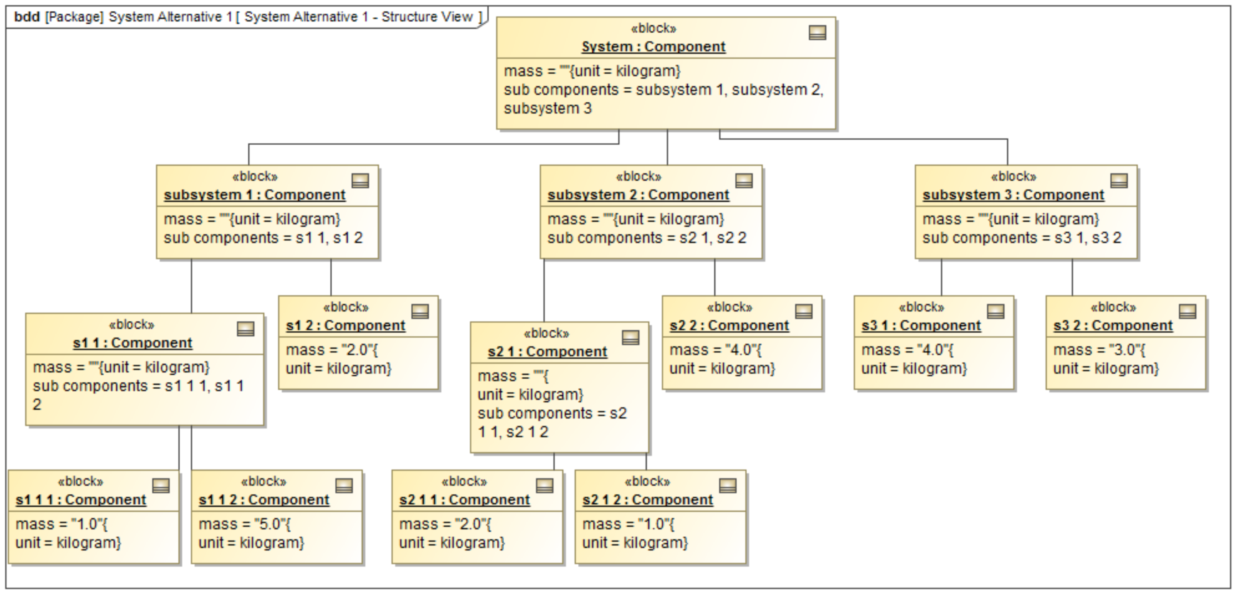
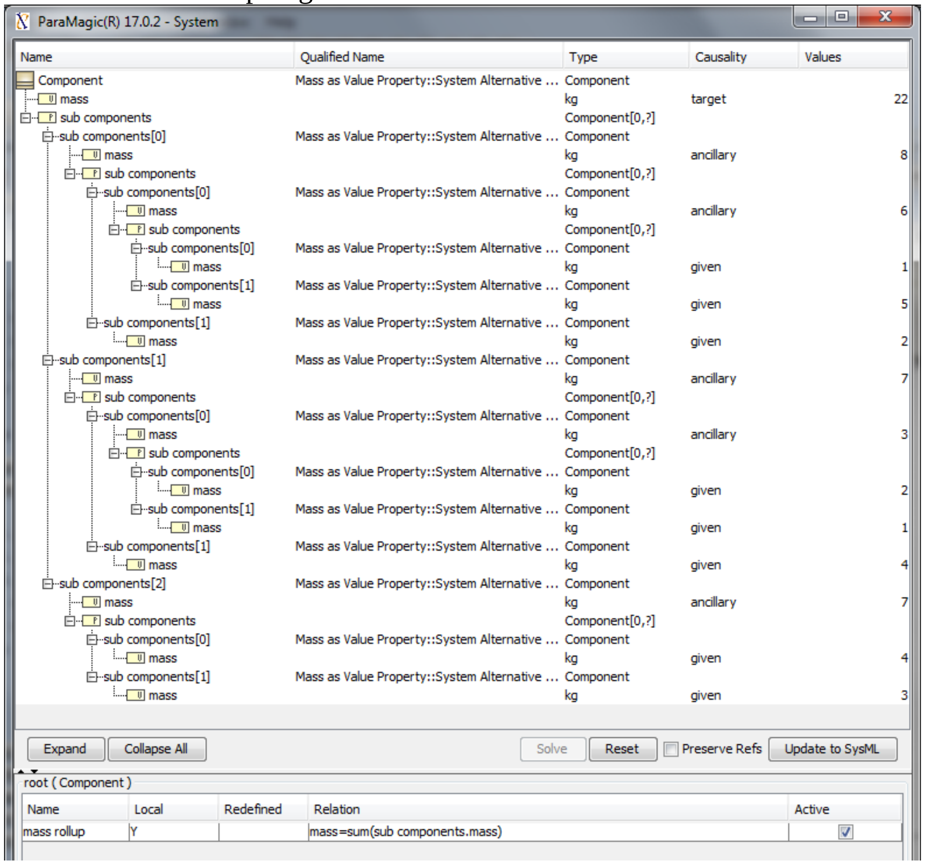

bdd [Package] System Alternative 1 [System Alternative 1 - Structure View]
«block»
System : Component
mass = ""{unit = kilogram}
sub components = subsystem 1, subsystem 2, subsystem 3
«block»
subsystem 1 : Component
mass = ""{unit = kilogram}
sub components = s1 1, s1 2
«block»
subsystem 2 : Component
mass = ""{unit = kilogram}
sub components = s2 1, s2 2
«block»
subsystem 3 : Component
mass = ""{unit = kilogram}
sub components = s3 1, s3 2
«block»
s1 1 : Component
mass = ""{unit = kilogram}
sub components = s1 1 1, s1 1 2
«block»
s1 2 : Component
mass = "2.0"{ unit = kilogram}
«block»
s2 1 : Component
mass = ""{ unit = kilogram}
sub components = s2 1 1, s2 1 2
«block»
s2 2 : Component
mass = "4.0"{ unit = kilogram}
«block»
s3 1 : Component
mass = "4.0"{ unit = kilogram}
«block»
s3 2 : Component
mass = "3.0"{ unit = kilogram}
«block»
s1 1 1 : Component
mass = "1.0"{ unit = kilogram}
«block»
s1 1 2 : Component
mass = "5.0"{ unit = kilogram}
«block»
s2 1 1 : Component
mass = "2.0"{ unit = kilogram}
«block»
s2 1 2 : Component
mass = "1.0"{ unit = kilogram}

Figure 14: Recursive parametric relationships can be executed in a recursive manner when ParaMagic® is launched at any instance in the hierarchy.

When ParaMagic® is launched for the top-level *System* instance, it executes the *mass rollup* recursive parametric relationship across all levels in the instance model as shown in Figure 15. The total mass of the System computes to 22 kgs.

Similarly, when ParaMagic® is launched for the *subsystem2* instance, it executes the *mass rollup* recursive parametric relationship across all components in the context of *subsystem2* (s2_1, s2_2, s2_1_1 and s2_1_2), as shown in Figure 16. The total mass of subsystem2 computes to 7 kgs.

Setup and execution of *Recursive Parametric Relationships* is a powerful feature of ParaMagic®. It provides a simple mechanism to model recursive constraints at the block level and execute them for system alternatives with various topologies at the instance level.

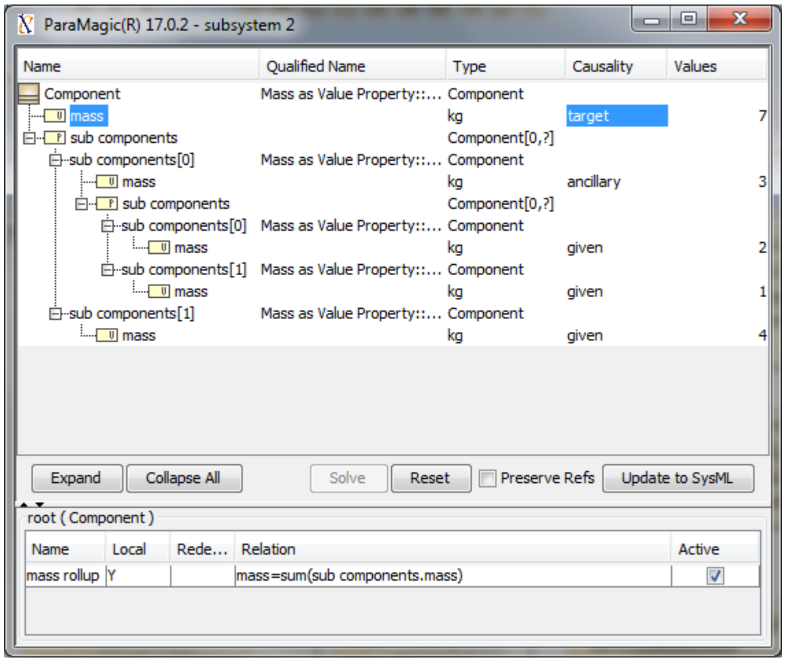


Name	Qualified Name	Type	Causality	Values
Component	Mass as Value Property::System Alternative ...	Component		
mass		kg	target	22
sub components		Component[0,?]		
sub components[0]	Mass as Value Property::System Alternative ...	Component		
mass		kg	ancillary	8
sub components		Component[0,?]		
sub components[0]	Mass as Value Property::System Alternative ...	Component		
mass		kg	ancillary	6
sub components		Component[0,?]		
sub components[0]	Mass as Value Property::System Alternative ...	Component		
mass		kg	given	1
sub components[1]	Mass as Value Property::System Alternative ...	Component		
mass		kg	given	5
sub components[1]	Mass as Value Property::System Alternative ...	Component		
mass		kg	given	2
sub components[1]	Mass as Value Property::System Alternative ...	Component		
mass		kg	ancillary	7
sub components		Component[0,?]		
sub components[0]	Mass as Value Property::System Alternative ...	Component		
mass		kg	ancillary	3
sub components		Component[0,?]		
sub components[0]	Mass as Value Property::System Alternative ...	Component		
mass		kg	given	2
sub components[1]	Mass as Value Property::System Alternative ...	Component		
mass		kg	given	1
sub components[1]	Mass as Value Property::System Alternative ...	Component		
mass		kg	given	4
sub components[2]	Mass as Value Property::System Alternative ...	Component		
mass		kg	ancillary	7
sub components		Component[0,?]		
sub components[0]	Mass as Value Property::System Alternative ...	Component		
mass		kg	given	4
sub components[1]	Mass as Value Property::System Alternative ...	Component		
mass		kg	given	3

root (Component)

Name	Local	Redefined	Relation	Active
mass rollup	Y		mass=sum(sub components.mass)	✓

Figure 15: Executing recursive parametric relationship (mass rollup) for the System instance



Name	Qualified Name	Type	Causality	Values
Component	Mass as Value Property::...	Component		
mass		kg	target	7
sub components		Component[0,?]		
sub components[0]	Mass as Value Property::...	Component		
mass		kg	ancillary	3
sub components		Component[0,?]		
sub components[0]	Mass as Value Property::...	Component		
mass		kg	given	2
sub components[1]	Mass as Value Property::...	Component		
mass		kg	given	1
sub components[1]	Mass as Value Property::...	Component		
mass		kg	given	4

root (Component)

Name	Local	Rede...	Relation	Active
mass rollup	Y		mass=sum(sub components.mass)	✓

Figure 16: Executing recursive parametric relationship (mass rollup) for the subsystem 2 instance

Copyright © 2014 InterCAX LLC. All Rights Reserved.

6.9.2 Redefinition

When building models of complex systems, *generic* concepts are successively specialized into several *concrete* concepts using inheritance. It is then required that these concrete concepts redefine properties and constraints inherited from generic concepts. Redefinition in SysML is the mechanism to achieve this. ParaMagic® provides the ability to detect and execute parametric relationships involving block properties—such as value, part, reference, shared, or constraint—that have been redefined, either implicitly or explicitly.

In ***explicit redefinition***, the owned properties of a block redefine inherited properties by referencing them in the SysML model—populating the *redefined property* field in the specification for an owned property. The owned property and the inherited property can have different names. See section 6.9.2.1 for details.

In ***implicit redefinition***, the owned property of a block redefines an inherited property by having the same name as the inherited property. See section 6.9.2.2 for details.

6.9.2.1 Explicit Redefinition

Explicit redefinition is the recommended approach for redefining inherited properties in SysML. The model shown in Figure 17 (BDD) illustrates explicit redefinition.

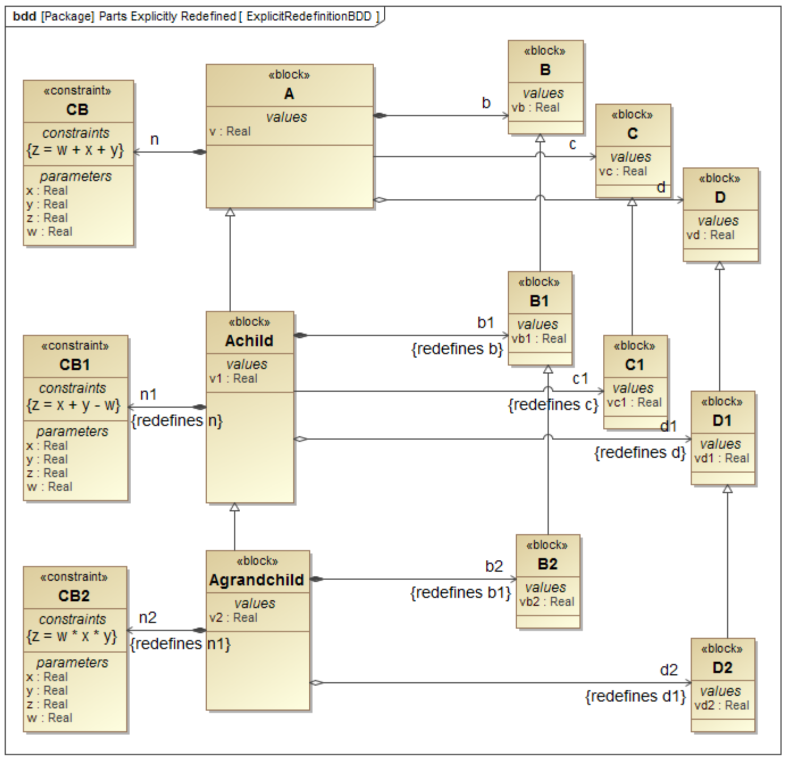


Figure 17: Explicit redefinition of inherited value, part, reference, shared, and constraint properties

Block A has the following properties: b (part property), c (reference property), d (shared property), n (contraint property), and v (value property). The parametric model for block A (top left in Figure 18) shows how the value properties owned by block A and its part/reference/shared properties are constrained (A.v = b.vb + c.vc + d.vd).

Block Achild is a subtype of block A and hence inherits all the properties of block A. However, Achild has its own part, reference, shared, constraint, and value properties that **explicitly redefine** the corresponding inherited properties. To achieve this, the *Redefined Property* field for each of the owned properties of Achild (b1, c1, d1, n1, and v1) was populated with the corresponding inherited property (b, c, d, n, and v respectively), as shown in Figure 19. As a result, the owned properties show the "redefine" keyword in Figure 17. The parametric model for Achild in Figure 18 (see top-right) shows how the value properties owned by block Achild and its part/reference/shared properties are constrained (Achild.v1 = c1.vc1 + d1.vd1 – b1.vb1). *Note how the constraint specification in the redefined constraint property (n1) is different compared to the constraint specification in the inherited constraint property (n).*

Similarly, the owned properties of block Agrandchild redefine the inherited properties b1, d1, n1. The inherited property c1 is not redefined and hence used by Agrandchild. The parametric model for Agrandchild in Figure 18 (bottom) shows how the properties of Agrandchild (redefined and inherited) are constrained (Agrandchild.v2 = b2.vb2 * c1.vc1 * d2.vd2). *Note how the constraint specification in the redefined constraint property (n2) is different compared to the constraint specification in the inherited constraint property (n1).*

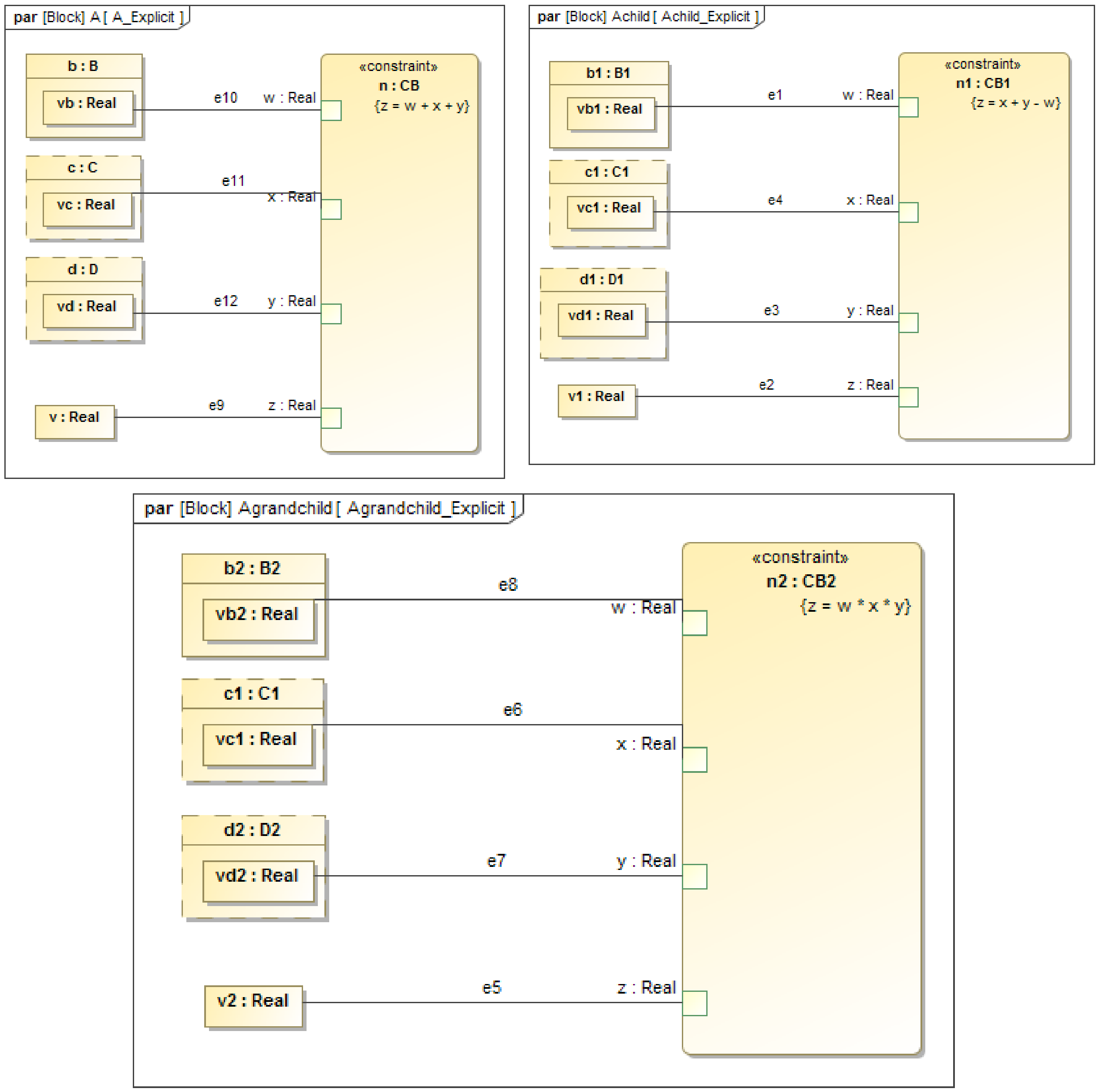
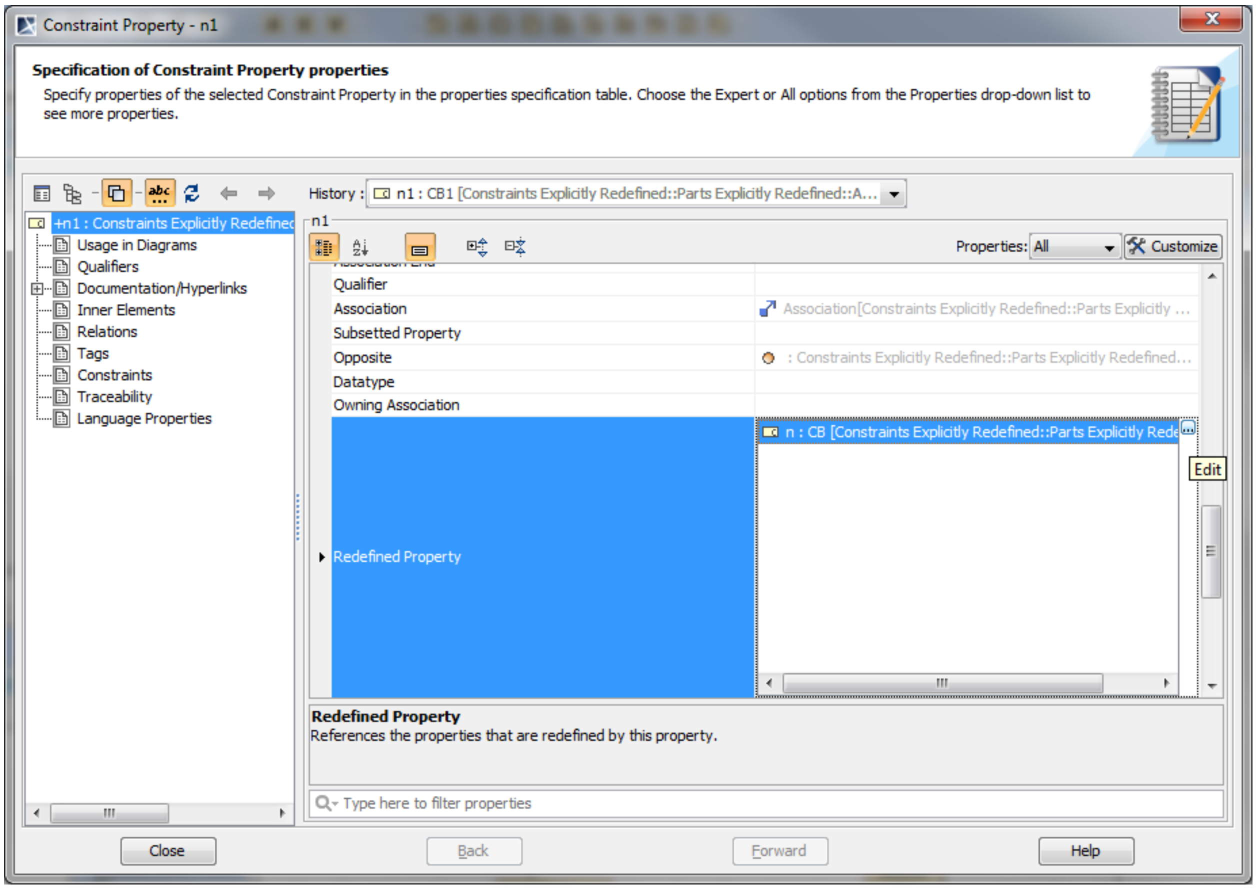


Figure 18: Parametric models for blocks A, Achild, and Agrandchild illustrating constraint relationships involving owned properties that explicitly redefine inherited properties

Copyright © 2014 InterCAX LLC. All Rights Reserved.



Constraint Property - n1

Specification of Constraint Property properties

Specify properties of the selected Constraint Property in the properties specification table. Choose the Expert or All options from the Properties drop-down list to see more properties.

History : n1 : CB1 [Constraints Explicitly Redefined::Parts Explicitly Redefined::A...

- +n1 : Constraints Explicitly Redefined
  - Usage in Diagrams
  - Qualifiers
  - Documentation/Hyperlinks
  - Inner Elements
  - Relations
  - Tags
  - Constraints
  - Traceability
  - Language Properties

n1

Properties: All

Qualifier	
Association	Association[Constraints Explicitly Redefined::Parts Explicitly ...
Subsetted Property	
Opposite	: Constraints Explicitly Redefined::Parts Explicitly Redefined...
Datatype	
Owning Association	
Redefined Property	n : CB [Constraints Explicitly Redefined::Parts Explicitly Rede

Redefined Property
References the properties that are redefined by this property.

Type here to filter properties

Figure 19: Explicitly redefining properties in MagicDraw

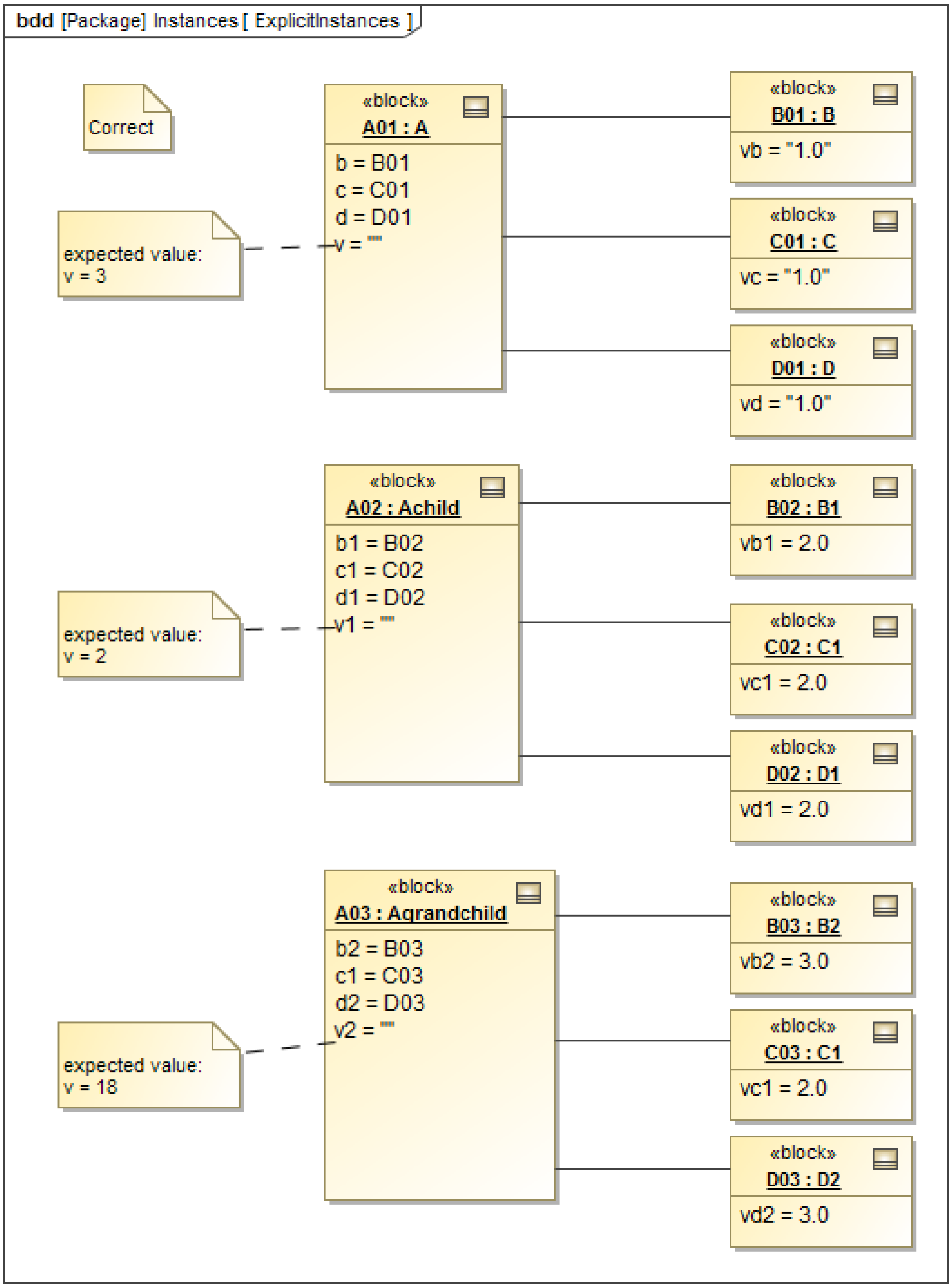
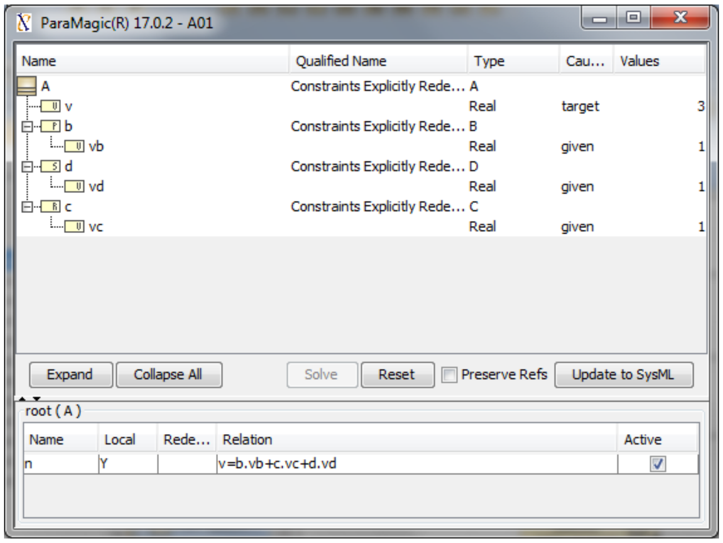


Figure 20: Instance model in ready-to-solve state

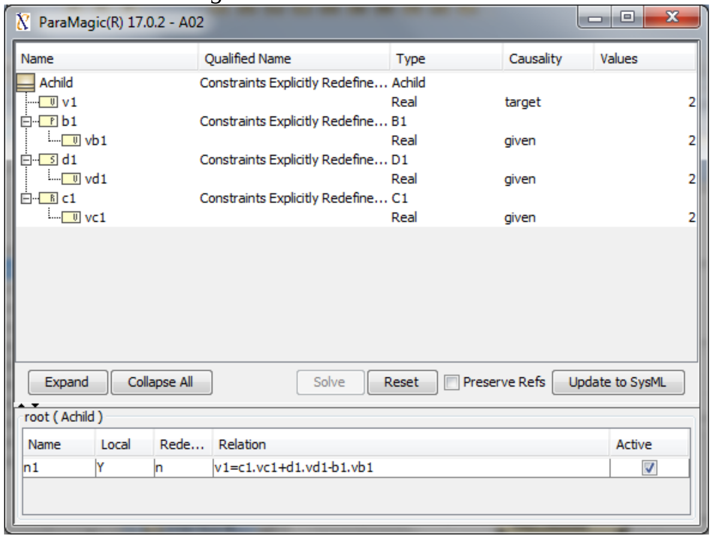
Copyright © 2014 InterCAX LLC. All Rights Reserved.



Name	Qualified Name	Type	Cau...	Values
A	Constraints Explicitly Rede...	A		
v		Real	target	3
b	Constraints Explicitly Rede...	B		
vb		Real	given	1
d	Constraints Explicitly Rede...	D		
vd		Real	given	1
c	Constraints Explicitly Rede...	C		
vc		Real	given	1

Name	Local	Rede...	Relation	Active
n	Y		v=b.vb+c.vc+d.vd	✓

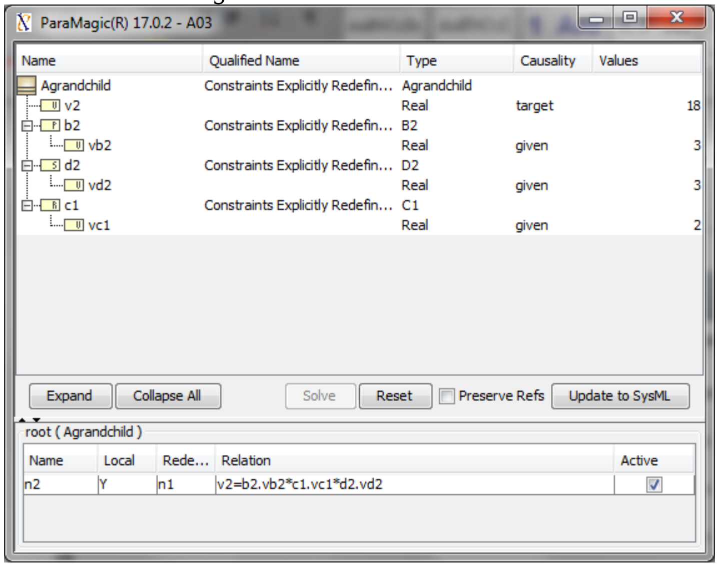
Figure 21: Solved instance A01



Name	Qualified Name	Type	Causality	Values
Achild	Constraints Explicitly Redefine...	Achild		
v1		Real	target	2
b1	Constraints Explicitly Redefine...	B1		
vb1		Real	given	2
d1	Constraints Explicitly Redefine...	D1		
vd1		Real	given	2
c1	Constraints Explicitly Redefine...	C1		
vc1		Real	given	2

Name	Local	Rede...	Relation	Active
n1	Y	n	v1=c1.vc1+d1.vd1-b1.vb1	✓

Figure 22: Solved instance A02



Name	Qualified Name	Type	Causality	Values
Agrandchild	Constraints Explicitly Redefin...	Agrandchild		
v2		Real	target	18
b2	Constraints Explicitly Redefin...	B2		
vb2		Real	given	3
d2	Constraints Explicitly Redefin...	D2		
vd2		Real	given	3
c1	Constraints Explicitly Redefin...	C1		
vc1		Real	given	2

Name	Local	Rede...	Relation	Active
n2	Y	n1	v2=b2.vb2*c1.vc1*d2.vd2	✓

Figure 23: Solved instance A03

Figure 20 shows instances A01 (of block A), A02 (of block Achild), and A03 (of block Agrandchild) in a ready-to-solve state. The ParaMagic® browser shows the same instances in a solved state in Figure 21, Figure 22, and Figure 23 respectively. Note that the ParaMagic® browser shows the redefined constraint properties for instances A02 and A03—see the equation section (lower part) in the browser in Figure 22 and Figure 23. The target slots (A01.v, A02.v1, and A03.v2) in the ParaMagic® browsers show the result of executing the parametric models for instances A01, A02, and A03 respectively.

6.9.2.2 Implicit Redefinition

Implicit redefinition is an alternative approach to redefining (overriding) inherited properties. The model shown in Figure 24 (BDD) illustrates implicit redefinition.

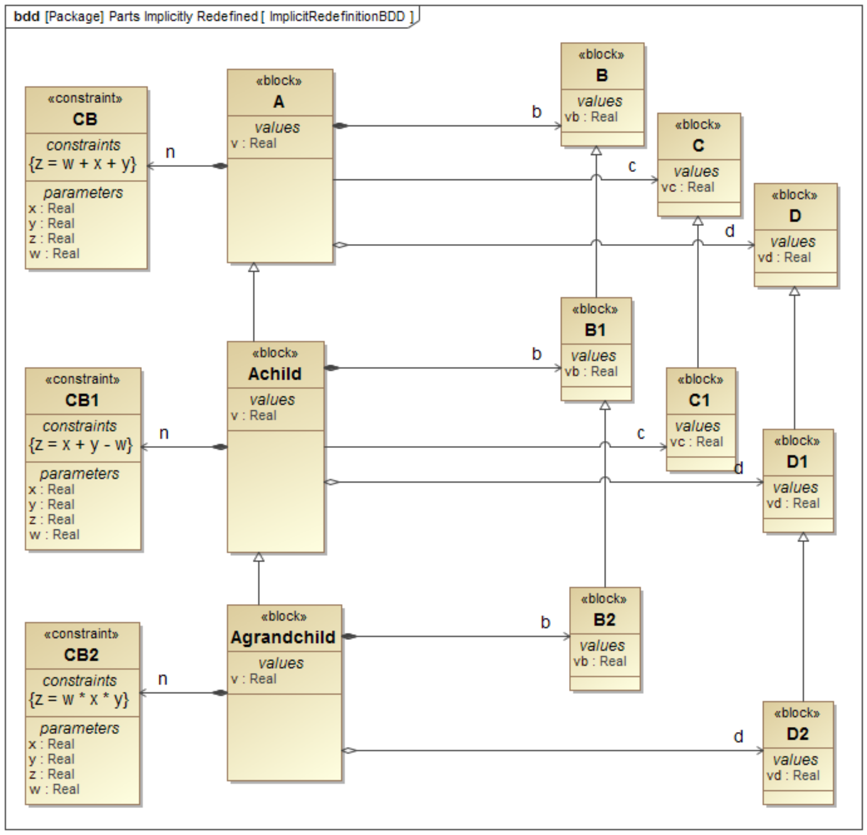


Figure 24: Implicit redefinition of inherited value, part, reference, shared, and constraint properties

This model is similar to the model shown in Figure 17 and described in the previous section except that Achild and Agrandchild have part, reference, shared, constraint, and value properties that override the inherited properties by having the same names. For example, Achild has an owned part property b that overrides the inherited part property (A.b) because it has the same name. The same pattern is used by Agrandchild for redefining properties inherited from Achild.

Though this method of using a name conflict to redefine inherited properties is easy to use since it does not require explicitly referencing the inherited property (as shown in Figure 19), it is ambiguous and not recommended due to poor model readability. However, ParaMagic® can treat name conflicts between owned and inherited properties as a form of redefinition and can execute models the same way as shown for explicit redefinition in the previous section.

Figure 25 illustrates the parameteric models for blocks A, Achild, and Agrandchild. The parametric models are setup in the same manner as those in Figure 18. The parametric models for blocks A and Achild have constraint relationships between their owned properties. The owned properties of block Achild implicitly redefine the properties inherited from block A because they have the same names. The parametric model for the block *Agrandchild* has a constraint relationship (*n*) between its owned properties (*Agrandchild.b*, *Agrandchild.d*, and *Agrandchild.n*) and an inherited property (*Achild.c*).

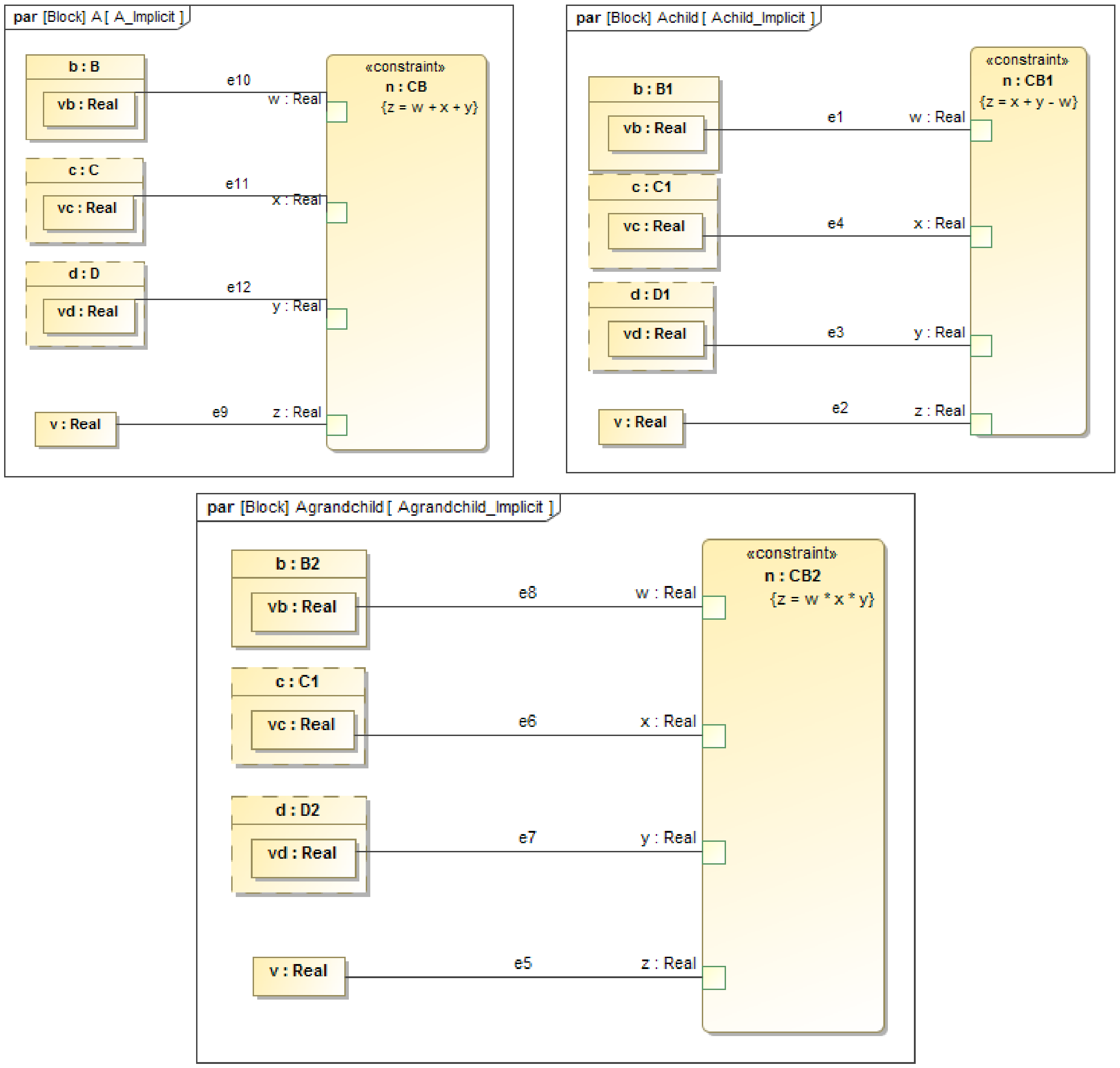


Figure 25: Parametric models for blocks A, Achild, and Agrandchild illustrating constraint relationships involving owned properties that implicitly redefine inherited properties

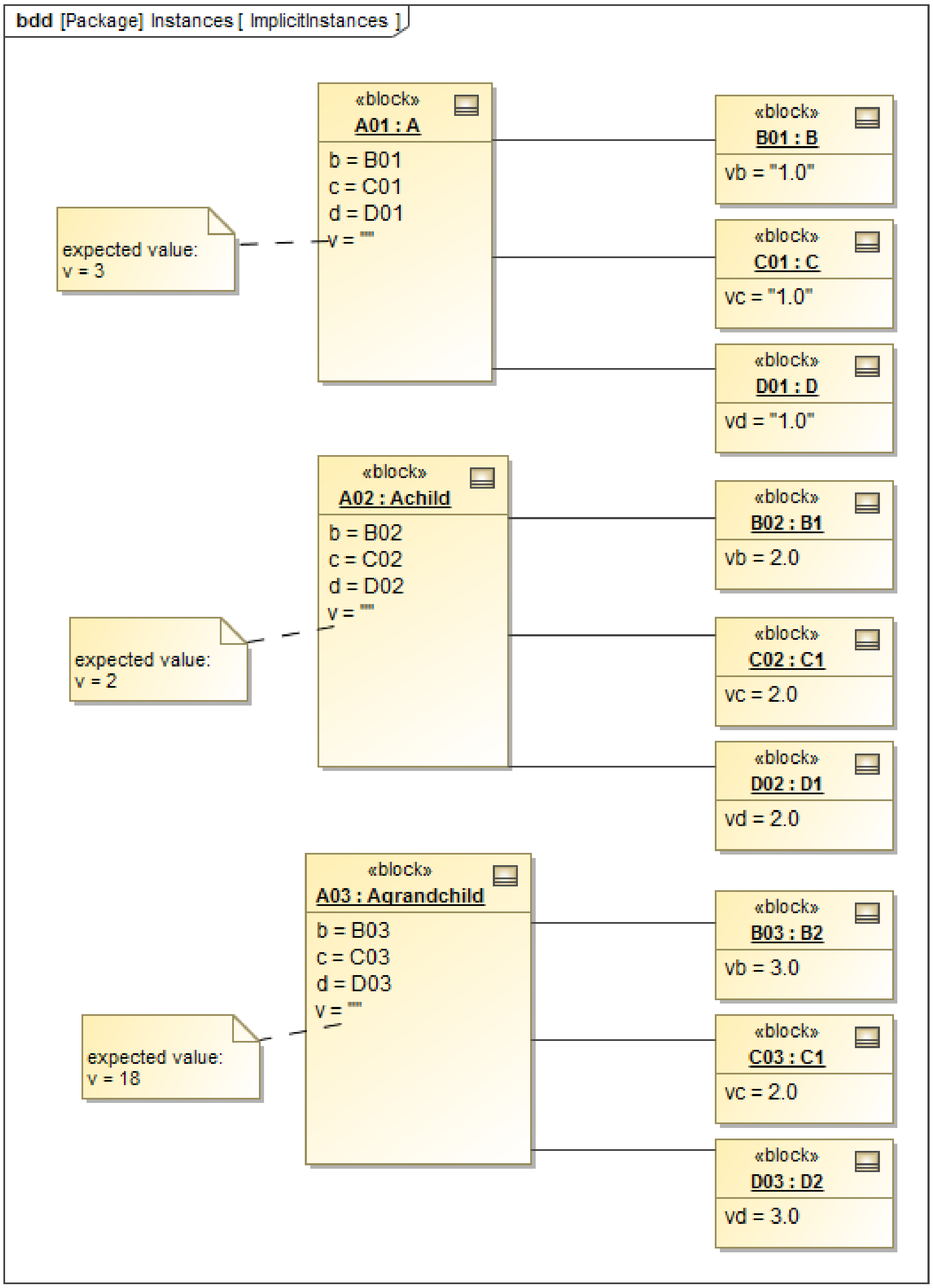
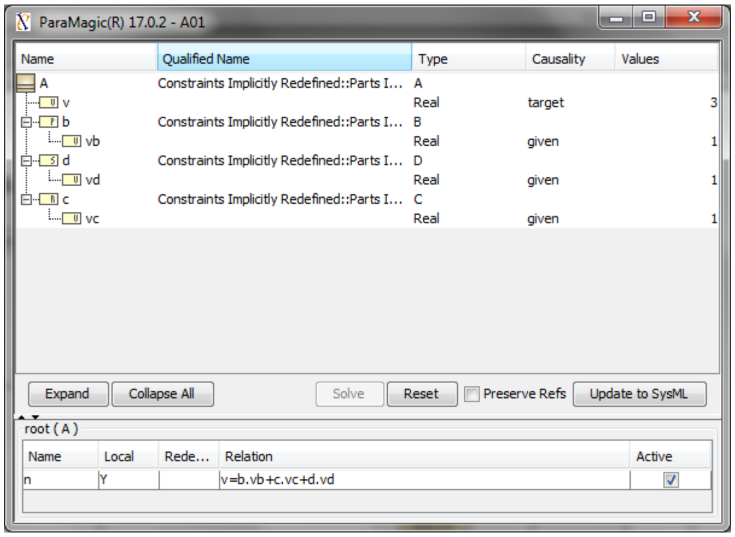


Figure 26: Instance model in ready-to-solve state

Copyright © 2014 InterCAX LLC. All Rights Reserved.

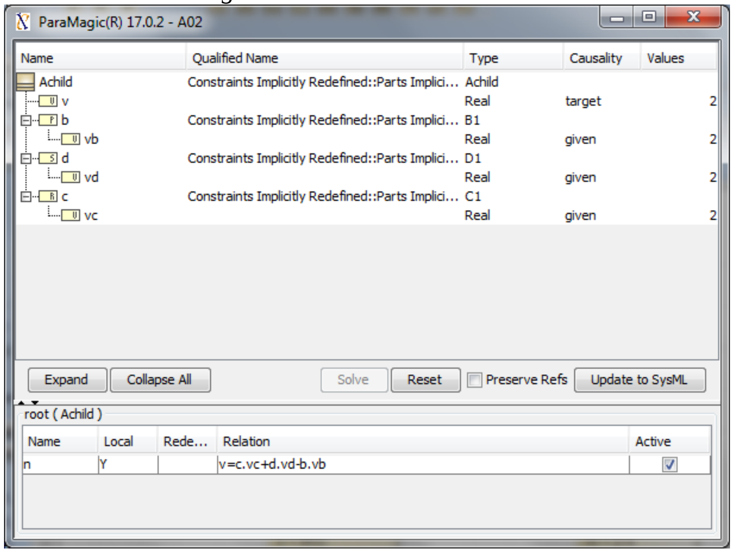
Figure 26 shows instances A01 (of blocs A), A02 (of block Achild), and A03 (of block Agrandchild) in a ready-to-solve state. The ParaMagic® browser shows the same instances in a solved state in Figure 27, Figure 28, and Figure 29 respectively. Note that the ParaMagic® browser shows the redefined constraint properties for instances A02 and A03 in the equation section (bottom part) of Figure 28 and Figure 29 respectively. The target slots (A01.v, A02.v, and A03.v) in the three ParaMagic® browsers show the result of executing the parametric models for instances A01, A02, and A03.



Name	Qualified Name	Type	Causality	Values
A	Constraints Implicitly Redefined::Parts I…	A		
v		Real	target	3
b	Constraints Implicitly Redefined::Parts I…	B		
vb		Real	given	1
d	Constraints Implicitly Redefined::Parts I…	D		
vd		Real	given	1
c	Constraints Implicitly Redefined::Parts I…	C		
vc		Real	given	1

Name	Local	Rede…	Relation	Active
n	Y		v=b.vb+c.vc+d.vd	✓

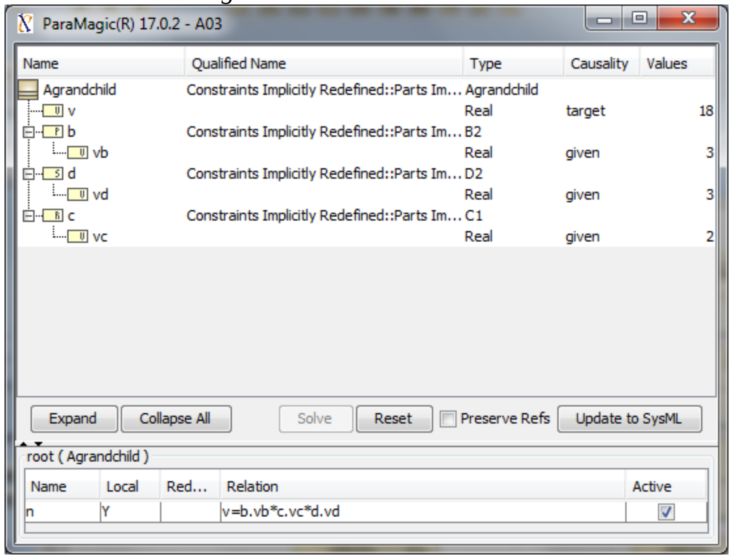
Figure 27: Solved instance A01



Name	Qualified Name	Type	Causality	Values
Achild	Constraints Implicitly Redefined::Parts Implici…	Achild		
v		Real	target	2
b	Constraints Implicitly Redefined::Parts Implici…	B1		
vb		Real	given	2
d	Constraints Implicitly Redefined::Parts Implici…	D1		
vd		Real	given	2
c	Constraints Implicitly Redefined::Parts Implici…	C1		
vc		Real	given	2

Name	Local	Rede…	Relation	Active
n	Y		v=c.vc+d.vd-b.vb	✓

Figure 28: Solved instance A02



Name	Qualified Name	Type	Causality	Values
Agrandchild	Constraints Implicitly Redefined::Parts Im…	Agrandchild		
v		Real	target	18
b	Constraints Implicitly Redefined::Parts Im…	B2		
vb		Real	given	3
d	Constraints Implicitly Redefined::Parts Im…	D2		
vd		Real	given	3
c	Constraints Implicitly Redefined::Parts Im…	C1		
vc		Real	given	2

Name	Local	Red…	Relation	Active
n	Y		v=b.vb*c.vc*d.vd	✓

Figure 29: Solved instance A03

6.10 Limitations

ParaMagic® 18.0 has the following limitations with respect to SysML models. Several of these are limitations of the core solvers.

1) OpenModelica (OM) as the core solver – If OM is selected as the core solver, the following limitations exist:
   a) For OM, the number of parametric relations should be equal to the number of unknown variables (value properties).
   b) Some math functions are not supported or have a limited support if OM is selected as the core solver. These functions are commented in blue in sections 6.5, 6.6, and 0.
   c) ParaMagic – Custom Mathematica Connection (section 8.1) is not supported
   d) ParaMagic – MATLAB Connection (section 8.3) is not supported

2) MATLAB Symbolic Math Toolbox (SMT) as the core solver – If MATLAB SMT is selected as the core solver, the following limitations exist:
   a) For SMT, the number of parametric relations should be equal to the number of unknown variables (value properties).
   b) Some math functions have a limited support as commented in red in sections 6.5-6.7.
   c) Conditional expressions do not work reliably when combined with other expressions. For e.g. **a = b + if(d>e, 1, 0)** will not work with MATLAB SMT. In these cases, users are requested to use multiple simpler expressions instead, for e.g. **a = b + c** and **c = if(d>e,1,0)**
   d) Inputs to *min* and *max* functions should be givens before solving. The functions do not work if their inputs are computed by solving other equations simultaneously.
   e) ParaMagic – Custom Mathematica Connection (section 8.1) is not supported

3) Constraint block-related limitations - This version of ParaMagic® does not support:
   a) multiple constraint specifications in a constraint block.
   b) constraint blocks composed of other constraint blocks.
   c) inequality constraints.

4) Binding connectors between constraint parameters – This version of ParaMagic® supports binding connectors between constraint parameters only when the multiplicity on constraint parameters is unspecified (1) or set to 1. If the multiplicity of one (or both) constraint parameter(s) at the ends of a binding connector is greater than 1, ParaMagic® shows the following validation error and lists the specific binding connectors. *As a workaround, users can create an intermediate value property and connect the constraint parameters to that intermediate value property*.

   The following constraint parameters used are connected by a binding connector, but one (or both) of them have multiplicity > 1. Currently, only binding connectors between constraint parameters with multiplicity 1 (or unspecified) are supported. As a workaround, create an intermediate value property with multiplicity > 1 and create binding connectors to that value property from both the constraint parameters. See the User Guide for further details.
   - Block "Many_Links::Domain2"
   "add1.c" and "sub1.a"
   "sub1.c" and "mult1.a"
   "div1.a" and "mult1.c"

5) Cyclic references-related limitations - This version of ParaMagic® does not support some types of cyclic references among model instance elements. Specifically, the following scenarios are not supported:

a) A block instance has a slot that it is populated with the instance itself, i.e. the instance points to itself.
b) Given two block instances, each has a slot that is populated with the other instance. For e.g. instance A points to instance B, and instance B points to instance A.

Note that a block may have a property typed by the block itself. This looping structure at the block level is supported if there is no looping at the instance level.

6) Value Types, Value Specification, and Causality
   a) See section 6.4 for requirements on value properties that can participate in parametric relations.
   b) ParaMagic® treats all numbers as real numbers.
   c) Values populating instance slots must be one of the following types: LiteralString, LiteralReal, or LiteralInteger.
   d) Slots corresponding to value properties that are typed by Integer (or its subtype) should have causality "given", i.e. they can only be inputs to parametric calculations. ParaMagic® treats integers (only allowed as inputs) as real numbers during solving.
   e) ParaMagic® does not perform automated unit conversions in math expressions. Thus, instance validation and solution do not check for the consistency of units. Users must provide instance values with consistent units.

7) If the value property corresponding to a slot is connected to the outputs of multiple ONEWAY relations, then the slot must have causality "given". This restriction is imposed to prevent instance models from being over-constrained—conflicting values of such slots may be computed from the different ONEWAY relations.

8) Complex numbers – This version of ParaMagic® does not support complex numbers. If solution results return a complex number for a variable, a string **No Value** is displayed for that variable in the ParaMagic® browser.

9) Complex Aggregate Relationships – Complex aggregate relationships with depth > 1 are not supported in this version of ParaMagic®. See section 6.8 for details.

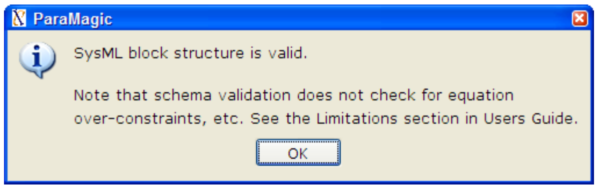
10) This version of ParaMagic® does not support scientific notation for numbers.

11) If you change the causality of a solved **target** variable to **given** in MagicDraw and launch the ParaMagic® browser, you may see an over-constrained set of values. ParaMagic® does not check for over-constrained instance values during validation. To resolve such states, users should identify a new target variable and re-solve the model.

12) When you successfully validate the schema structure of a model (see **Validate** in section 7.1), you will receive the message in Figure 30.
    a) The warning is to inform you that this validation function effectively checks the syntax and connectivity of the structure of your SysML model schema as far as parametrics solving goes.
    b) At this stage ParaMagic® does not check for over-constrained equations and similar non-syntactic issues. In general the math solver that you can later invoke via the ParaMagic® Browser will uncover such issues (usually by returning a "No Value" result or an over-constraint warning.

13) When you successfully validate the instance structure of a model (see **Validate** in section 7.1), you will receive the message in Figure 31.
    a) The warning is to inform you that this validation function effectively checks the syntax and connectivity of the structure of your SysML model instance as far as parametrics solving goes.

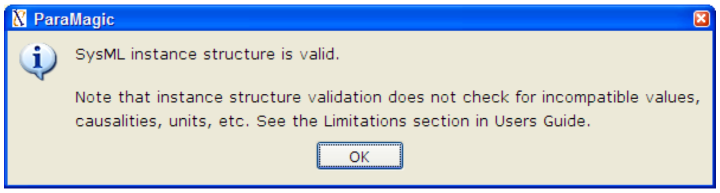
b) ParaMagic® does not check for over-constrained values or inconsistent causalities and similar non-syntactic issues at the instance validation stage. Such issues will be uncovered during the instance solution stage, usually by returning a **No Value** result (in the ParaMagic® browser) or an over-constraint warning.



SysML block structure is valid.

Note that schema validation does not check for equation over-constraints, etc. See the Limitations section in Users Guide.

Figure 30: Structure validation notice



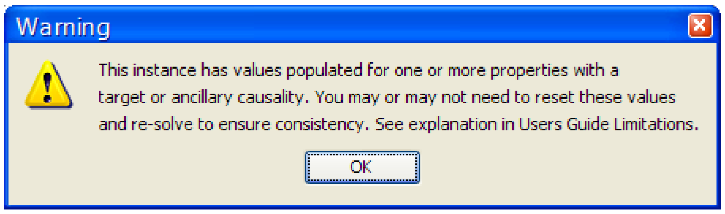
SysML instance structure is valid.

Note that instance structure validation does not check for incompatible values, causalities, units, etc. See the Limitations section in Users Guide.

Figure 31: Instance validation notice

14) When you open an existing instance from MagicDraw for browsing (see **Browse** in section 7.1) you will see the warning message in Figure 32 if that instance has values for any attribute with causality "target" or "ancillary".
   a) This situation exists because you have stored previously solved values in the SysML model. Usually it is not a problem and you can browse the instance to review its results without concern (if you fully control the model and know its change history), *but in general we recommend re-setting the instance and re-solving it to be safe*.
   b) There are several ways that previously solved instance values can become invalid (because the instance in MagicDraw is not continuously connected to the instance in the ParaMagic® browser), including:
      i) Someone changed the corresponding schema structure in MagicDraw after that instance was solved (e.g., they added a new equation or changed an existing equation).
      ii) Someone changed an attribute value or an attribute causality in the instance in MagicDraw. In general we recommend not making such changes in MagicDraw for a previously solved instance. Instead, open the ParaMagic® browser and then make such changes.

   In other words, for example if you make structural changes to your SysML model schema, such as redrawing the connectors in a parametric diagram, you can still browse the old solved instance (conforming to the schema before changes) in ParaMagic®. With the changes in the model, the values in old instances may be non-conformant to the model schema. This version of ParaMagic® does not automatically re-solve (update) old instances to conform with the new schema. It is recommended that after structural changes to the model schema, users should (a) re-validate the schema, (b) open the ParaMagic® browser, (c) Reset all non-given values (automatically by pressing the **Reset** button), and (d) solve the instance again.



This instance has values populated for one or more properties with a target or ancillary causality. You may or may not need to reset these values and re-solve to ensure consistency. See explanation in Users Guide Limitations.

Figure 32: Warning regarding previously solved instance values.

15) Mathematica solving issues:
   a) If a system of equations is under-constrained, ParaMagic® browser shows that no value is available for one or more target variables after solving. It does not explicitly warn the user that the system of equations was under-constrained.

b) In some cases, while solving a large system of equations in the reverse direction (non-natural direction), Mathematica returns an over-constrained set of values (without exceptions or warnings). Contact us (*support@intercax.com*) for further information.

16) ParaMagic®-MATLAB Connection: When constraint blocks (properties) wrapping MATLAB functions/scripts are connected with value properties that have multiplicity > 1, then the size of array returned by MATLAB must match the size of array initialized in the SysML instance model. Specifically,

a) If the size of array returned from MATLAB > size of array in the SysML instance model, then ParaMagic® throws an error message as below.

The number of results ("50") obtained from the external model do not match with the number of allowable values ("1") in the expression "outs=xfwExternal(matlab,function,matlab_fmpmo,ins1)" in "Bugs::Output Mismatch::Instance01::Instance01".

b) If the size of array returned from MATLAB < size of array in the SysML instance model, then ParaMagic® solves the model but shows the following warning

WARN The number of results ("50") obtained from the external model do not match with the number of allowable values ("51") in the expression "outs=xfwExternal(matlab,function,matlab_fmpmo,ins1)" in "Bugs::Output Mismatch::Instance02::Instance02". The first "50" values are used as the result.

c) If the size of array returned from MATLAB = size of array in the SysML instance model, then ParaMagic® solves the model.

17) Plotting: The plotting functions available with ParaMagic® use the ParaMagic-Custom Mathematica connection (section 8.1.3). This requires that users select Mathematica or PlayerPro as their core solver. During parametric model execution, users may be prompted to select a Math link program.

a) **For Windows**, if asked to select a Mathlink program, select **WolframPlayerPro.exe** in the PlayerPro installation (if using PlayerPro), or select **Mathematica.exe** in the Mathematica installation (if using Mathematica).
b) **For Mac**, users will NOT be asked to select a Mathlink program. Plots will be created if Mathematica or PlayerPro is selected as the core solver.
c) **For Linux**, users will NOT be asked to select a Mathlink program. Plots will be only created if the symbolic link **/usr/local/bin/math** is pointing to Mathematica math kernel **/usr/local/Wolfram/Mathematica/9.0/Executables/math**

7 Program Features

7.1 Command Menus

ParaMagic® commands are applied by right-mouse-clicking on the appropriate block or package in the MagicDraw containment tree and selecting the menu item **ParaMagic**.

Command	Description

Validate[18]	When applied to a **block**, it validates the block structure. Block structure refers to the block itself and all its properties recursively (e.g. parts of parts).
Browse	• When applied to an **instance**, it validates the instance structure and launches the ParaMagic® browser for that instance structure. Here instance structure refers to the instance itself and all its slots recursively. • When applied to a **package** for which a default instance has been set, it validates the default instance and launches the ParaMagic® browser (same as above).
Util > **Assign default causalities**	• When applied to an **instance**, it assigns default causality to each variable (slot) in the instance structure. Here instance structure refers to the instance itself and all its slots recursively. Variables that have value(s) are assigned causality *given*, and variables without value(s) are assigned causality *undefined*. At least one variable must be assigned manually as *target* to solve the model. • When applied to a **package**, it assigns default causality to all instances (and their related instances) in the package.
Util > **Set default instance**	When applied to an **instance**, it assigns that instance as the default instance for the package owning that instance. Once assigned, users can browse the instance (launch the ParaMagic® browser) from the package.
Util > Convert Connector to Binding Connector	When applied to a connector (listed in Relations under a SysML Block), it converts the connector to a binding connector. When using MagicDraw, it is possible for users to accidently create connectors instead of binding connectors on a parametric diagram. ParaMagic® only reads binding connectors in a parametric model as per the SysML specification. This utility helps users to upgrade a connector to a binding connector.
Util > Silent > Solve	When applied to an instance, it solves the parametric model in the context of that instance, and writes out the results to the ParaMagic® Silent Target log file (see section 7.4) without launching the ParaMagic® browser. This action is the same as launching the ParaMagic® browser (**Browse** command) on an instance and then pressing the *Solve* button.
Util > Silent > Solve and Update	When applied to an instance, it has the same behavior as **Util > Silent > Solve** command (above) plus it also updates the SysML instance model with the solved results. This action is the same as launching the ParaMagic® browser (**Browse** command) on an instance, pressing the *Solve* button, and then pressing the *Update to SysML* button after solving completes.
Excel > Setup	When applied to an **instance**, it launches the Excel setup utility to connect instances to Excel spreadsheets. This connection is used to read/write values from/to spreadsheets to/from SysML instance models. See section 8.2.2 for details.
Excel > Read from Excel	When launched on a **slot/instance/package**, it reads values from connected Excel spreadsheet(s) into the instance slots. See the table at the end of Step 6 in section 8.2.2.1.
Excel > Write to Excel	When launched on a **slot/instance/package**, it writes values from instance

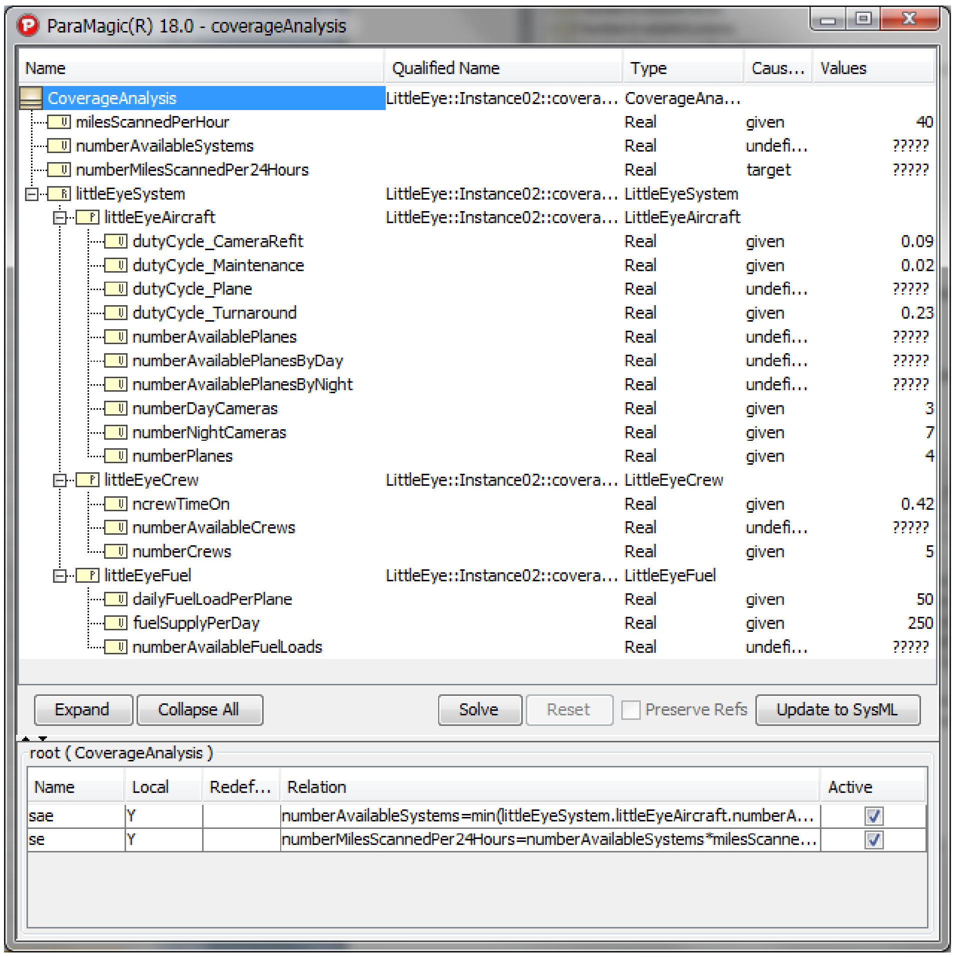
[18] Note that ParaMagic® validation may not include validation errors / warnings in the SysML model that are shown by MagicDraw. It is recommended that users resolve validation errors shown by MagicDraw before using ParaMagic® capabilities. ParaMagic® further validates the block structure, instance structure, and parametric model to ensure that it is solvable.

	slots to the connected Excel spreadsheets. See the table at the end of Step 6 in section 8.2.2.1.
Excel > Create Instances from Excel	When applied to a **block**, it launches ParaMagic® instance generation utility which is used to generate instances of the block based on tabulated data in Excel spreadsheet(s). See section 8.2.1 for details.
Trade Study > Setup	• When applied to an **instance**, it launches the trade study setup utility. • When applied to a package for which a default instance has been set, it launches the trade study setup utility for the default instance.
Trade Study > Run	• When applied to an **instance**, it runs the trade study • When applied to a package for which a default instance has been set, it runs the trade study for that default instance.
Help > Users Guide	When applied to **any model element**, it launches the ParaMagic® User Guide, located at **<MD_Root>\manual\ParaMagic Plugin UserGuide.pdf**. The User Guide is also accessible via **MagicDraw Help > Other Documentation > ParaMagic Plugin UserGuide**
Help > Tutorials	When applied to **any model element**, it launches the ParaMagic® tutorials folder, located at **<MD_Root>\samples\ParaMagic\Tutorials**

7.2 Browser

The ParaMagic® plugin browser displays the parametric model variables and controls for solving and displaying the variable values. An example of the browser window is shown in Figure 33 below. The ParaMagic® browser displays properties in the following order—value, part, shared, and reference.

The ParaMagic® browser only displays slots whose corresponding value properties are typed by Real, Integer, String, or a value type representing a quantity, and their subtypes. For more details, refer to section 6.4.



Name	Qualified Name	Type	Caus...	Values
CoverageAnalysis	LittleEye::Instance02::covera...	CoverageAna...		
milesScannedPerHour		Real	given	40
numberAvailableSystems		Real	undefi...	?????
numberMilesScannedPer24Hours		Real	target	?????
littleEyeSystem	LittleEye::Instance02::covera...	LittleEyeSystem		
littleEyeAircraft	LittleEye::Instance02::covera...	LittleEyeAircraft		
dutyCycle_CameraRefit		Real	given	0.09
dutyCycle_Maintenance		Real	given	0.02
dutyCycle_Plane		Real	undefi...	?????
dutyCycle_Turnaround		Real	given	0.23
numberAvailablePlanes		Real	undefi...	?????
numberAvailablePlanesByDay		Real	undefi...	?????
numberAvailablePlanesByNight		Real	undefi...	?????
numberDayCameras		Real	given	3
numberNightCameras		Real	given	7
numberPlanes		Real	given	4
littleEyeCrew	LittleEye::Instance02::covera...	LittleEyeCrew		
ncrewTimeOn		Real	given	0.42
numberAvailableCrews		Real	undefi...	?????
numberCrews		Real	given	5
littleEyeFuel	LittleEye::Instance02::covera...	LittleEyeFuel		
dailyFuelLoadPerPlane		Real	given	50
fuelSupplyPerDay		Real	given	250
numberAvailableFuelLoads		Real	undefi...	?????

root (CoverageAnalysis)

Name	Local	Redef...	Relation	Active
sae	Y		numberAvailableSystems=min(littleEyeSystem.littleEyeAircraft.numberA...	✓
se	Y		numberMilesScannedPer24Hours=numberAvailableSystems*milesScanne...	✓

Variable Browser (shown in expanded form)

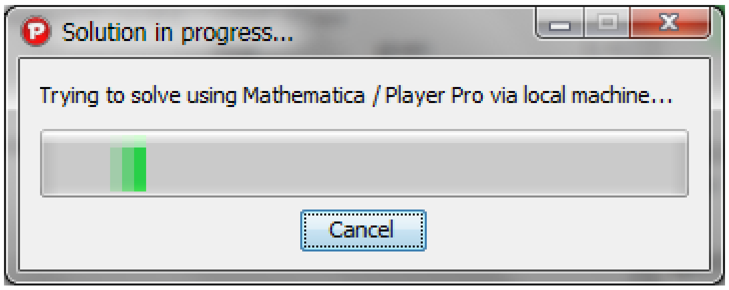
Toolbar

Relationship Browser

Figure 33: Browser window

7.2.1 "Solution in progress" Window

When the Solve button on the browser is clicked, a network of parametric equations is constructed and solved using Mathematica / PlayerPro / OpenModelica / MATLAB SMT. ParaMagic® waits for the results to be returned. During this interval, a **Solution in progress** window is displayed, as shown in Figure 34below.



Trying to solve using Mathematica / Player Pro via local machine...

Figure 34: Solution in progress window when using Mathematica / PlayerPro

The **Cancel** button causes ParaMagic® to ignore any intermediate results returned from the solvers, to reset the model values (same as pushing **Reset**), and to return to the ready-to-solve condition. It does not necessarily affect the solver directly (i.e., the solver job may continue to execute, in which case

ParaMagic® will ignore any results it obtains). If the solver job itself hangs, it may require manual intervention before you can continue solving models via ParaMagic®.

7.2.2 Variable Browser

Each variable is shown with Name, Type, Causality and Value. Variable causality can be changed in the browser window by clicking on the causality type and selecting from the list. Four possibilities are available for Causality. A user can only specify causality to be **Given**, **Undefined**, or **Target**. If a variable with initial causality **Undefined** is used to compute other variables with during solution, its causality will automatically change to **Ancillary** after solving.

Command	Description
Given	Value is assigned before solving
Undefined	Value may be calculated during solving if it is needed to determine a **Target**, directly or indirectly.
Target	Value is calculated during solving (if Mathematica can find a valid solution). At least one unknown must be assigned as a target variable to initiate the solution process.
Ancillary	A variable whose value is calculated during solving and used to calculate the value of another variable. The value of this variable is not available before solving.

The value of a **Given** variable can be changed by clicking on its value and editing it. All other variables must be changed to Given before they can be edited (see 7.2.5 for more about this).

The size of the Variable Browser window may be expanded by 1) dragging down the lower border of the Browser window, followed by 2) dragging down the horizontal black line between the Toolbar and Relationship Browser.

7.2.3 Toolbar

Command	Description
Expand	Expands all blocks in the variable browser one tree level per button click
Collapse All	Fully collapses the tree structure of the variable browser
Reset	Resets the values of all target and ancillary variables in a solved instance, and changes the causality of ancillary variables to undefined
Solve	Exports the model to Mathematica for execution and displays the results in the Browser window after solving is complete
Update to SysML	Causes the results in the Browser window to be exported to the SysML instance in MagicDraw

7.2.4 Relationship Browser

The Relationship Browser displays the constraint equations present in the parametric model and shows their current status during solution. By selecting one of the blocks in the Variable Browser (e.g. LittleAircraft in Figure 33), the equations specific to that part of the model are displayed in the Relationship Browser.

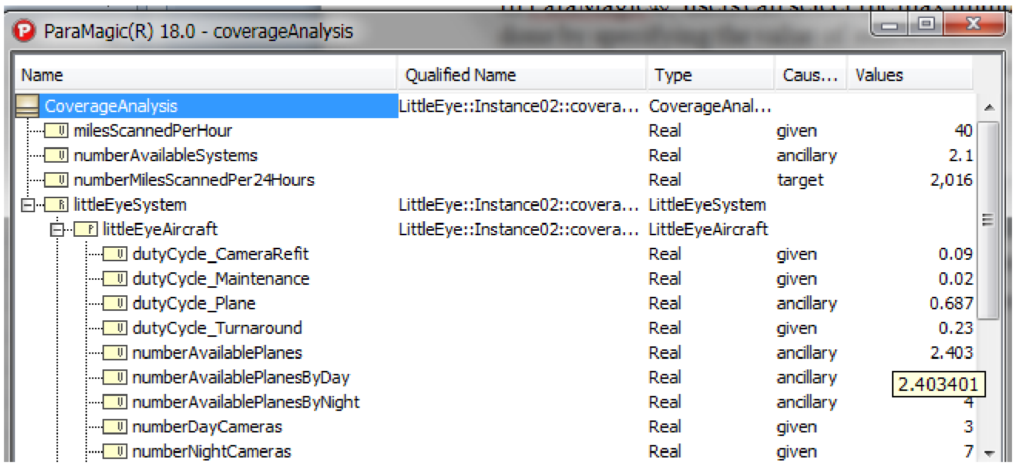
The checkbox in the Active column allows individual equations to be “turned off” during solving, i.e. not exported to Mathematica. This may prevent other parts of the parametric model from being solved.

The columns Local and Oneway refer to local vs. inherited characteristics and static causality characteristics of the equations. These are not user-controlled via this browser—they are determined by the structure of the model, which the user can change in MagicDraw and then re-open in the ParaMagic® browser to see the updated Local or Oneway properties.

7.2.5 Editing an Instance in the Browser Window

Attribute values and attribute causalities can be modified within the Browser window, as well as in the SysML instance diagram before launching the browser. Note that editing the SysML instance after the Browser is launched will not update the Browser. Changing either a value or causality in the Browser after a solution has been calculated will return the model to an unsolved state, which can then be solved with the changed parameters.

In ParaMagic®, users can select the max number of decimal places to display in the browser. This can be done by specifying the value of **maxNumberOfDecimalsToDisplay** variable in the ParaMagic.ini file. See section 7.3 below for details. When a user points the mouse cursor over the displayed value, the browser will indicate the true value of the variable. If a user double-clicks on value with causality= "**given**", the true value will be displayed in the edit box.

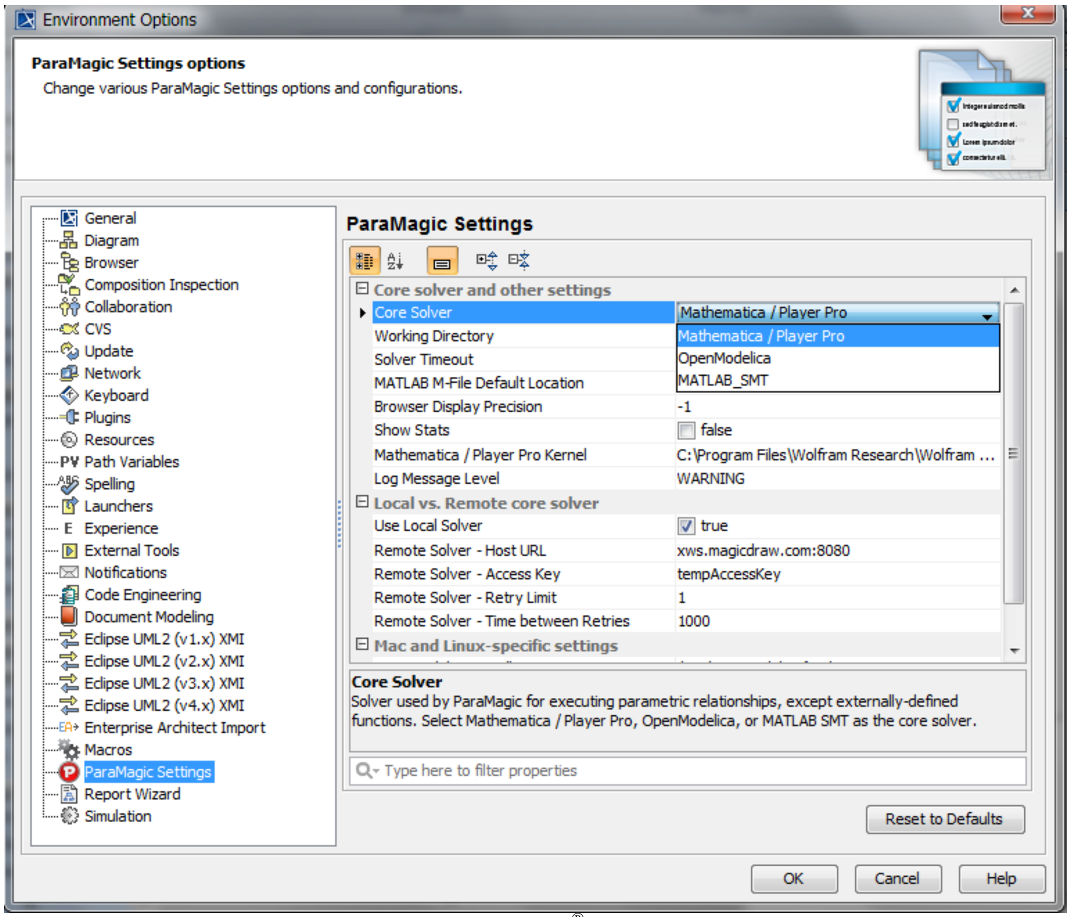

ParaMagic(R) 18.0 - coverageAnalysis

Name	Qualified Name	Type	Caus...	Values
CoverageAnalysis	LittleEye::Instance02::covera...	CoverageAnal...		
milesScannedPerHour		Real	given	40
numberAvailableSystems		Real	ancillary	2.1
numberMilesScannedPer24Hours		Real	target	2,016
littleEyeSystem	LittleEye::Instance02::covera...	LittleEyeSystem		
littleEyeAircraft	LittleEye::Instance02::covera...	LittleEyeAircraft		
dutyCycle_CameraRefit		Real	given	0.09
dutyCycle_Maintenance		Real	given	0.02
dutyCycle_Plane		Real	ancillary	0.687
dutyCycle_Turnaround		Real	given	0.23
numberAvailablePlanes		Real	ancillary	2.403
numberAvailablePlanesByDay		Real	ancillary	2.403401
numberAvailablePlanesByNight		Real	ancillary	4
numberDayCameras		Real	given	3
numberNightCameras		Real	given	7

Figure 35: Hover the mouse pointer over the displayed value to see the true value.

7.3 ParaMagic® Settings

The ParaMagic® Settings menu provides the capability to set variables that control ParaMagic® behavior. The ParaMagic® Settings menu can be launched from MagicDraw by selecting **Options > Environment**, and then selecting **ParaMagic Settings**, as shown in Figure 36 below.



ParaMagic Settings options

Change various ParaMagic Settings options and configurations.

ParaMagic Settings

Core solver and other settings	
Core Solver	Mathematica / Player Pro
Working Directory	Mathematica / Player Pro
Solver Timeout	OpenModelica
MATLAB M-File Default Location	MATLAB_SMT
Browser Display Precision	-1
Show Stats	false
Mathematica / Player Pro Kernel	C:\Program Files\Wolfram Research\Wolfram ...
Log Message Level	WARNING
Local vs. Remote core solver	
Use Local Solver	✓ true
Remote Solver - Host URL	xws.magicdraw.com:8080
Remote Solver - Access Key	tempAccessKey
Remote Solver - Retry Limit	1
Remote Solver - Time between Retries	1000
Mac and Linux-specific settings	

Core Solver
Solver used by ParaMagic for executing parametric relationships, except externally-defined functions. Select Mathematica / Player Pro, OpenModelica, or MATLAB SMT as the core solver.

Figure 36: ParaMagic® Settings Menu

If you select a variable you will see a short description of that variable, as shown in Figure 36 above. The function of each setting/variable is described below.

Core solver and other settings

1. **Core Solver** – This variable is used to select the core solver. Options are Mathematica / PlayerPro, OpenModelica, or MATLAB Symbolic Math Toolbox (SMT).

2. **Working Directory** – This variable is used to specify the working directory for ParaMagic®. This directory is used by ParaMagic® to save temporary files during the solving process. If the field is empty, ParaMagic® uses the default location which is a temp folder insider the ParaMagic® Working Folder (see section 7.4), such as the following on Windows 7.
   <Your user dir>\AppData\Local\.magicdraw\18.0\plugins\com.intercax.paramagic\temp

   Note: If you are using OpenModelica as the core solver, ensure that the working directory field does not have any spaces.

3. **Solver Timeout** - This variable is used to specify the time interval in seconds—measured after pressing the Solve button in the ParaMagic® browser—after which ParaMagic® will disconnect from the external solver (e.g. Mathematica / PlayerPro / MATLAB). Note that this will not kill the solver runtime process. However, ParaMagic® will not wait for the solution results. By default, the value of this variable is set to -1 which indicates no timeout.

4. **MATLAB M-File Default Location -** This variable is used to specify the default location of MATLAB M-files (functions or scripts) when using the ParaMagic-MATLAB Connection capability (see section 8.3). Though the location of MATLAB M-files can be specified for each constraint block wrapping the file, it is sometimes useful to specify a default location. ParaMagic® checks if the location of the M-file is specified in the constraint block. If not, ParaMagic® will check for the M-file in the default location specified here.

5. **Browser Display Precision -** This variable is used to specify the max number of decimal places to display for values in the ParaMagic® browser. If the default value (-1) is specified, location-specific default settings will be used, for e.g. displaying 3 decimal places in US.

6. **Show Stats –** When this variable is set to true, ParaMagic® will show statistics about the parametric model (during validation/execution) in the MagicDraw message window.

7. **Mathematica / PlayerPro Kernel** - This variable is used to specify the location of the MathKernel executable file in Mathematica or Player Pro installation. This variable is used only for Windows and Mac operating systems. For Linux, a symbolic link **/usr/local/bin/math** must exist that links to the math kernel available with PlayerPro or Mathematica installation. Refer to the Installation section (Step 3a in section 4.2) for details on setting this variable for Windows, Mac, and Linux. This variable is used only if **Core Solver** variable is set to Mathematica / Player Pro and if **Use Local Solver** variable is set to true.

8. **Log Message Level -** This setting allows users to control the verbosity of log messages displayed in the MagicDraw Message (Console) Window when using ParaMagic®. The default verbosity is WARNING, i.e. all messages with severity of warning or above (such as error) will be shown.

Local vs. Remote core solver

9. **Use Local Solver -** This variable is used to specify if ParaMagic™ should use local or remote solver. In the current version of ParaMagic®, this setting applies only to Mathematica. If set to true, ParaMagic® will use Mathematica installed on the user's machine. If set to false, ParaMagic® will use the Mathematica server available remotely.

10. **Remote Solver – Host URL** –This variable is used to specify the location of remote Mathematica server (if *Use Local Solver* variable is set to *true* above). Values are specified in the following format: **IP number (or alias): port number**.

11. **Remote Solver – Access Key -** This variable is used to specify the access key for the remote Mathematica server (if *Use Local Solver* variable is set to *true* above).

12. **Remote Solver – Retry Limit -** This variable is used to specify the number of times ParaMagic® will retry connecting to the remote Mathematica server if it is unable to connect to the server in the first try (e.g. due to network issues). This value is used only if *Use Local Solver* is set to *true*.

13. **Remote Solver – Time between Retries –** This variable is used to specify the time interval (in milliseconds) after which ParaMagic® will retry connecting to the Mathematica server if it is unable to connect to the server in the first try (e.g. due to network issues). This value is used only if *Use Local Solver* is set to *true*.

Copyright © 2014 InterCAX LLC. All Rights Reserved.

Mac and Linux-specific settings – These settings are applicable only when using ParaMagic® on Mac or Linux operating systems.

14. **OpenModelica Installation Location** – This variable is used to indicate the location of OpenModelica (omc file) on Mac OS X, for e.g. **/opt/openmodelica/bin/omc**. This variable is used only if the core solver is set to OpenModelica. This variable does not need to be set for Linux.

15. **MATLAB Installation Location** – This variable is used to set the location of MATLAB executable (matlab file) on Mac or Linux, e.g. **/Applications/MATLAB_R2014a.app/bin/matlab**. If you are using MATLAB on 32-bit Mac platform, then append the location with " –maci", e.g. **/Applications/MATLAB_R2014a.app/bin/matlab -maci**

7.4 ParaMagic® Working Folder

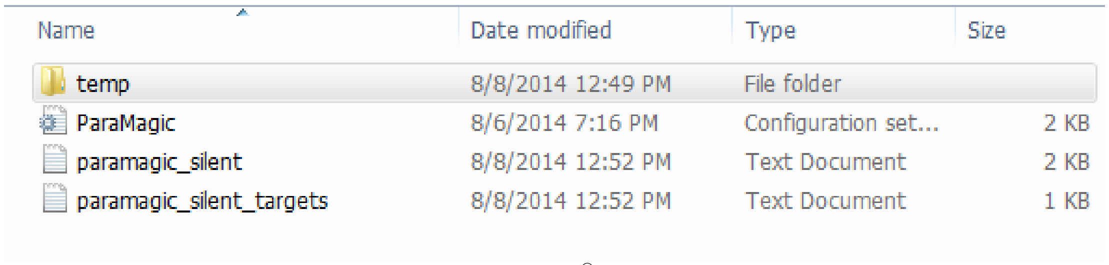
During operation, ParaMagic® saves working files for models in the same location as MagicDraw working folder. Example locations for Windows and Mac are provided below.

- On Windows 7, this location is:
<Your user dir>\AppData\Local\.magicdraw\18.0\plugins\com.intercax.paramagic, such as **C:\Users\InterCAX\AppData\Local\.magicdraw\18.0\plugins\com.intercax.paramagic**

- On Windows XP, this location is:
C:\Documents and Settings\<Your user dir>\Local Settings\Application Data\.magicdraw\17.0.5\plugins\com.intercax.paramagic

- On Mac OS X, this location is:
<Your user dir>/.magicdraw/18.0/plugins/com.intercax.paramagic, such as **/Users/InterCAX/.magicdraw/18.0/plugins/com.intercax.paramagic**

The ParaMagic® Working Folder includes a **temp** folder that stores files created by ParaMagic® when solving parametric models. It also includes the **paramagic_silent** and **paramagic_silent_targets** log files that are created when using ParaMagic® Silent feature (section 10). The **ParaMagic.ini** file located here should <u>never</u> be edited manually.



Name	Date modified	Type	Size
temp	8/8/2014 12:49 PM	File folder	
ParaMagic	8/6/2014 7:16 PM	Configuration set...	2 KB
paramagic_silent	8/8/2014 12:52 PM	Text Document	2 KB
paramagic_silent_targets	8/8/2014 12:52 PM	Text Document	1 KB

Figure 37: ParaMagic® Working Folder

8 CONNECTIONS TO EXTERNAL TOOLS

8.1 ParaMagic - Custom Mathematica Connection

The ParaMagic - Custom Mathematica connection (a.k.a cMathematica) exposes full power of Mathematica to ParaMagic users through the cMathematica function. This allows "wrapping" of both pre-defined and user-written Mathematica functions—saved as **.m** files in the Mathematica Autoload folder—as constraint blocks. Nine pre-defined graphing functions and three statistical functions are included with ParaMagic™. The user can create as many custom functions as desired if he or she has familiarity with Mathematica function programming and access to the Mathematica directory structure. The canonical form of the constraint expression is:

output parameter = cMathematica(function name, input arguments)

and would be used in a parametric diagram as shown in Figure 38.

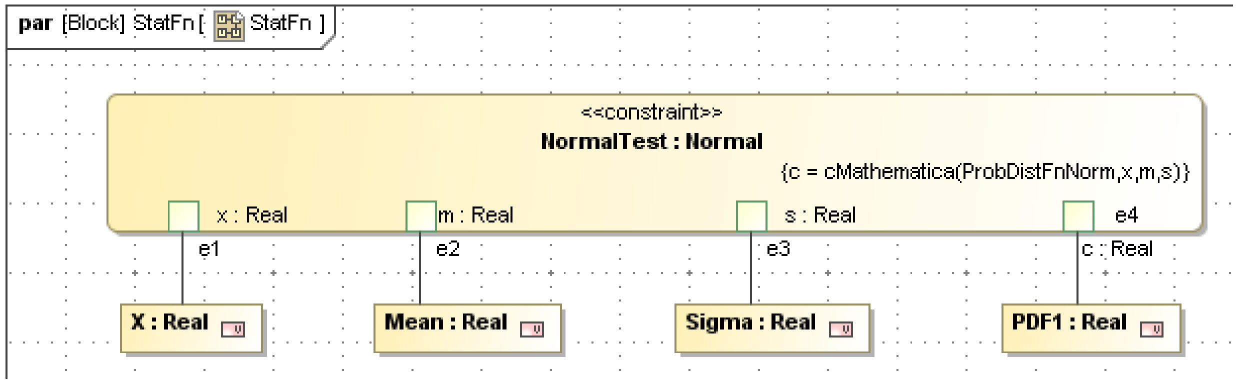


Figure 38: Constraint Expression using cMathematica Function in Parametric Diagram

The Orbital tutorial provided with ParaMagic™ demonstrates an example usage of a graphing function provided by cMathematica feature. It is located under **<MD_Root>\samples\ParaMagic\Tutorials\Orbital.**

This feature is not supported if OpenModelica or MATLAB SMT is selected as the core solver.

8.1.1 Installation

A library of pre-defined Mathematica graphing and statistical functions is provided with ParaMagic® 16.9. After ParaMagic™ installation, this library is located here: **<MD_Root>\plugins\com.intercax.paramagic\xfw\conf\ICAX.zip.** If you are using local Mathematica, right click on the **ICAX.zip** file and extract it to **<Your Mathematica Installation>\SystemFiles\Autoload** folder. For example, if you have local Mathematica version 7.0 on your Windows machine, extract **ICAX.zip** to **C:\Program Files\Wolfram Research\Mathematica\7.0\SystemFiles\Autoload.** If you are using XWS-based Mathematica, ask your system administrator to extract **ICAX.zip** to your Mathematica server installation. If you are evaluating ParaMagic™ and using our test server **xws.magicdraw.com**, then you do not need to follow the steps above. The library is already extracted in the Mathematica installation at **xws.magicdraw.com** and ready for you to evaluate.

Note: After extracting **ICAX.zip** file to **C:\Program Files\Wolfram Research\Mathematica\7.0\SystemFiles\Autoload** folder, verify that the **Autoload** folder has an **ICAX** folder and Mathematica **.m** files inside the **ICAX** folder. There should not be an **ICAX** folder inside the **ICAX** folder.

8.1.2 Usage

Tutorial 6 (Orbital) in the ParaMagic tutorials describes the process of using the pre-defined graphical and statistical Mathematica functions (sections 8.1.3 and 8.1.4 below). See section 6.2, Step IV, in ParaMagic Tutorials document for details. Note that the ParaMagic Constraint Block Library package—loaded with the ParaMagic Profile—has a generic cMathematica_Function constraint block that can be copied (in your package) and used in the manner described in Tutorial 6.

ParaMagic™ allows users to specify the location of the plots created by the Mathematica graphing functions. The location can be specified for each constraint block that wraps a Mathematica function. The tag **working_dir** (owned by the stereotype **External_Model**) is populated for the constraint specification in the constraint block to specify the location—also described in Tutorial 6. The behavior of ParaMagic with respect to the **working_dir** tag is as below:

1) If the tag **working_dir** is populated and
   a) the folder location exists, ParaMagic® will pass the location as the first argument to the Mathematica function.
   b) the folder location does not exist, ParaMagic® will show an error message to the user.

2) If the tag **working_dir** is not populated, ParaMagic™ will not pass the location to the Mathematica function. Therefore, the invoked Mathematica function should not expect its first argument to be the folder location.

Note that the **working_dir** tag is used to pass the location of a folder to a Mathematica function. This folder can be used for exporting plots or other forms of outputs (e.g. .txt or .csv files with the results).

Both input and output parameters of the constraint block wrapping the cMathematica function can be single-valued or multi-valued (e.g. an array of real number) but not a complex data structure (e.g. multi-dimensional arrays).

8.1.3 Graphing Functions

Nine pre-defined graphing functions have been created to provide quick access to some of the two-dimensional plotting functions in Mathematica. In the interests of simplicity, the flexibility of these pre-defined functions is limited. Users familiar with Mathematica function programming may want to create their own routines.

These nine functions have several common features:

- When executed, each function creates and saves a plot file with a fixed name to a location specified by the user. Calling the function a second time will over-write the first file. The location of the plot file is specified in the constraint block used in the model.
- Each function returns a single parameter (single value real) to ParaMagic® on completion of the function, with a value of 1.
- The input arguments can include numeric data, expressed as real single value or aggregate data, and strings which appear as labels on the plot. The strings cannot be entered as variables (constraint parameters); they must be explicitly fixed in the constraint specification.
- Blank spaces are not allowed in the names of the output and the input parameters of the cMathematica function.
- In the following table, Title will be displayed as the title of the graph, XAxis will be displayed as the horizontal axis label and YAxis will be displayed as vertical axis label. Strings must be enclosed in quotes, e.g. "Power_versus_Time". Since plot and axes labels are also input arguments to the cMathematica function, blank spaces are not allowed. For example, if the plot title is "Power versus Time", ParaMagic™ will throw an instance parsing error.
- It is frequently convenient to create a hyperlink in the MagicDraw model pointing to the output file created. For model portability, it is recommended to define the hyperlinked location relative to the MagicDraw installation. You may use <install.root> to refer to your MagicDraw installation. For example, **<install.root>\plugins\com.intercax.paramagic\xfw\conf\PlotXY.jpg** will automatically resolve to the plot file location.

Note: If you are using PlayerPro as the core solver, you will be prompted for a front-end program when solving parametric models that use these graphing functions. When prompted, select the WolframPlayerPro executable file on your computer. This is the main program located in your PlayerPro installation, e.g. **C:\Program Files\Wolfram Research\Wolfram Player Pro\9.0\WolframPlayerPro.exe**

Function	Output	Description
ICAXPlotX	PlotX.jpg	**Plots a line graph of the x values vs. {1,2,3,…}** **z =cMathematica(ICAXPlotX,x,"Title", "XAxis", "YAxis")**
ICAXPlotXT	LinePlotXT.jpg	**Plots a line graph of x vs. t values** **Z =cMathematica(ICAXPlotXT,t,x,"Title", "XAxis", "YAxis")**
ICAXPlotXY	LinePlotXY.jpg	**Plots a two line graph of x and y values vs. {1,2,3,…}** **Z =cMathematica(ICAXPlotXY,x,y,"Title", XAxis","YAxis")**
ICAXPlotXYT	LinePlotXYT.jpg	**Plots a two line graph of x and y values vs. t values** **z =cMathematica(ICAXPlotXYT,t,x,y,"Title", "XAxis", "YAxis")**
ICAXPlotXYScatter	ScatterPlotXY.jpg	Plots a scatter plot of y vs. x values **z =cMathematica(ICAXPlotXYScatter,x,y,"Title", "XAxis", "YAxis")**
ICAXBarChartX	BarChartX.jpg	Plots a bar chart of the x values vs. {1,2,3,…} **z =cMathematica(ICAXBarChartX,x,"Title", "XAxis", "YAxis")**
ICAXBarChartXY	BarChartXY.jpg	Plots a double bar chart of x and y values vs. {1,2,3,…} **z =cMathematica(ICAXBarChartXY,x,y,"Title", "XAxis", "YAxis")**
ICAXPieChartX	PieChartX.jpg	Plots a pie chart of the x values vs. {1,2,3,…} **z =cMathematica(ICAXPieChartX,x,"Title")**
ICAXHistogramX	HistogramChartX.jpg	Plots a histogram of the x values vs. {1,2,3,…} **z =cMathematica(ICAXHistogramX,x,"Title","XAxis", "YAxis")**

8.1.4 Statistical Functions

Three pre-defined statistical functions have been created to provide quick access to some of the statistical power of Mathematica.

Function	Description
ProbDistFnBinom	**Returns the probability distribution function for outcome k in a binomial distribution of n trials and success probability p (k and n integers)** **c =cMathematica(**ProbDistFnBinom**,n,p,k)**
ProbDistFnNorm	**Returns the probability distribution function for value x in a normal distribution with mean m and standard deviation s** **c =cMathematica(**ProbDistFnNorm**,x,m,s)**
ProbDistFnPois	**Returns the probability distribution function for value k in a Poisson distribution with mean m (k integer)** **c =cMathematica(**ProbDistFnPois**,m,k)**

8.1.5 User-Defined Mathematical Functions

A user familiar with Mathematica function programming can easily create and use custom Mathematica functions within SysML parametric diagrams with the cMathematica capability. There are two ways to creating custom functions, using one of the five preset UserfnN.m functions (where N runs from 1 to 5) or creating a new function using one of the existing pre-defined functions as a template for compatibility with ParaMagic.

8.1.6 UserfnN.m

Within the ICAX library with the pre-defined graphing and statistical functions, we have provided five "empty" functions. Each one can be edited using Mathematica or any standard text editor. Replace the comment lines

```
(* :Add Mathematica code calculating output b from inputs t and m *)
(* :To save graph, Export["GraphFileName", GraphFunction[arguments]] *)
```

with valid Mathematica code, which will be executed when the function is called.

Note that only two input arguments are defined in the function definition. This number can be reduced or increased with appropriate modification of the template. Output and input parameters can be single-valued or multi-valued (e.g. array).

8.1.7 Custom Functions

The user can write his/her own Mathematica functions using the cMathematica capability. We recommend using one of the pre-defined functions as a template to insure compatibility with ParaMagic.

In order for a function to be recognized by Mathematica, it must be autoloaded on start-up. See the Mathematica user documentation for discussion on declaring and loading functions. One easy way to accomplish this is to

- Save the new **.m** file in **<Mathematica installation directory>\SystemFiles\Autoload\ICAX**
- Edit the **Master.m** and **Kernel\init.m** files in the **ICAX** folder to declare the new function. Add the lines

  ```
  DeclarePackage["ICAX`NewFunctionName`", {"NewFunctionName"}]
  ```

  to each of these files and save, where **NewFunctionName** is the name of the new function, without the **.m** extension.

8.2 ParaMagic - Excel Connection

The ParaMagic - Excel Connection (PM-EC) allows users to perform two key functions:

- Generate / Update SysML instance model structure from Excel spreadsheets
- Read/Write instance slot values from/to Excel spreadsheets.

This enables users to quickly generate topologically variant system architectures (as instance models), perform parametric analyses on these architectures, and export results for plotting and presentation to spreadsheets. In section 8.2.1 the capability to generate instance models from Excel spreadsheets is presented, and in section 8.2.2 the capability read/write instance slot values from/to Excel spreadsheets is presented.

8.2.1 Generate/Update SysML Instance Model Structure from Excel Spreadsheets

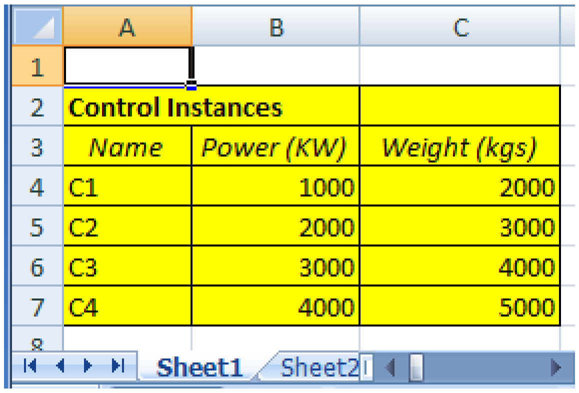
In this section, we present a step-by-step process for generating SysML instances from Excel spreadsheets. For simplicity, we will first describe the process for primitive blocks and then for complex blocks. A primitive block is one that has no part/reference/shared properties. These blocks represent the

Copyright © 2014 InterCAX LLC. All Rights Reserved.

leaf-level systems/parts in a system hierarchy. A complex block is one that has part/reference/shared properties.

8.2.1.1 *Creating instances of a primitive block*

1. Suppose you have a table of Control system instances (alternatives) in Excel spreadsheet as shown below. The columns represent properties of the Control system. This table may have been developed independently of the SysML model, so the names of properties (e.g. Power and Weight) may not correspond to the property names in the SysML model and the table may have more columns than required/relevant for SysML instances.

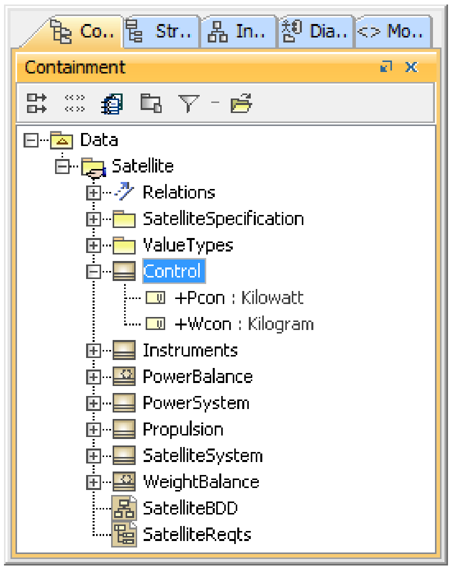


	A	B	C
1			
2	Control Instances		
3	Name	Power (KW)	Weight (kgs)
4	C1	1000	2000
5	C2	2000	3000
6	C3	3000	4000
7	C4	4000	5000

Sheet1
Sheet2

Figure 39: Example spreadsheet with instances of Control system

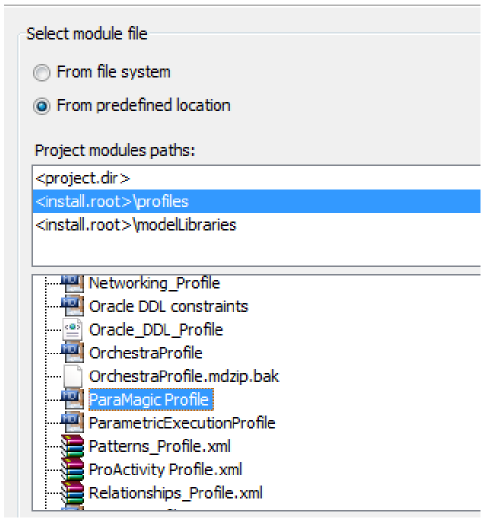
2. Now suppose you have a primitive block (e.g. Control) in the SysML model with value properties Pcon and Wcon. The block may have been defined in the SysML model or defined in a separate model that is imported as module (library), as shown below in Figure 40.



Data
Satellite
Relations
SatelliteSpecification
ValueTypes
Control
+Pcon : Kilowatt
+Wcon : Kilogram
Instruments
PowerBalance
PowerSystem
Propulsion
SatelliteSystem
WeightBalance
SatelliteBDD
SatelliteReqts

Figure 40: Control block in the SysML model

3. Import the ParaMagic profile into your model if it is not imported already. To import the profile, go to **File > Use Module** in MagicDraw, select <install.root>\profiles, select ParaMagic Profile (as shown below), and then press the Finish button.


Select module file

From file system

[✓ From predefined location]

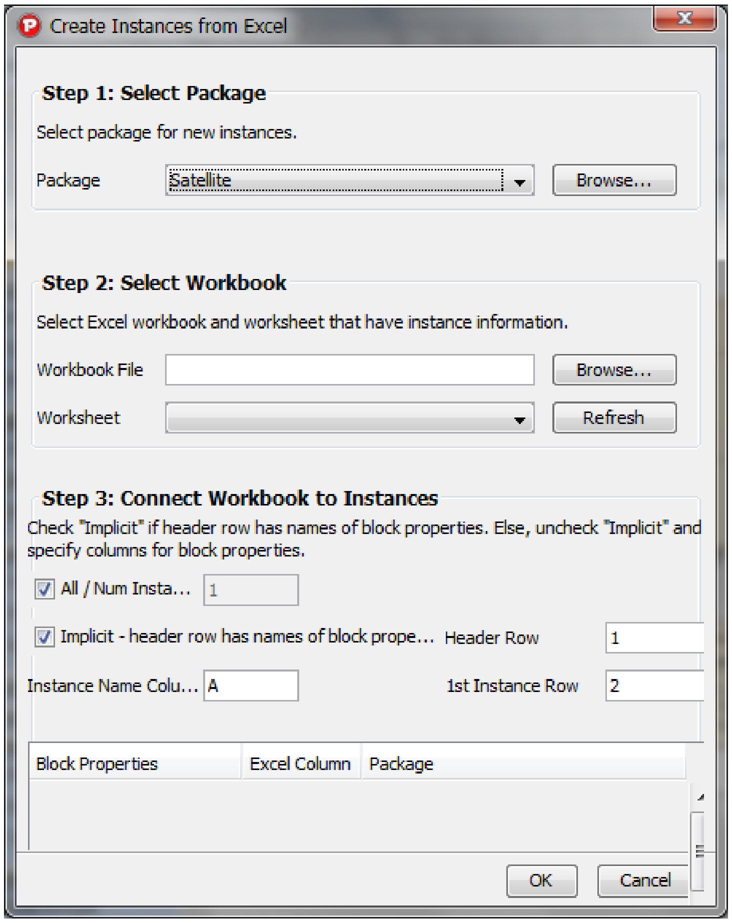
Project modules paths:

<project.dir>
<install.root>\profiles
<install.root>\modelLibraries

Networking_Profile
Oracle DDL constraints
Oracle_DDL_Profile
OrchestraProfile
OrchestraProfile.mdzip.bak
ParaMagic Profile
ParametricExecutionProfile
Patterns_Profile.xml
ProActivity Profile.xml
Relationships_Profile.xml

Figure 41: Import the ParaMagic Profile

4. To generate instances of the Control block from the table in the spreadsheet, right click on the Control block in the MagicDraw Containment tree and select **ParaMagic > Excel > Create Instances from Excel**. You will see the ParaMagic instance creation utility, as shown below.


Create Instances from Excel

Step 1: Select Package

Select package for new instances.

Package: Satellite

Step 2: Select Workbook

Select Excel workbook and worksheet that have instance information.

Workbook File:

Worksheet:

Step 3: Connect Workbook to Instances

Check "Implicit" if header row has names of block properties. Else, uncheck "Implicit" and specify columns for block properties.

[✓] All / Num Insta... 1

[✓] Implicit - header row has names of block prope... Header Row: 1

Instance Name Colu... A   1st Instance Row: 2

Block Properties	Excel Column	Package

Figure 42: ParaMagic instance creation utility

Copyright © 2014 InterCAX LLC. All Rights Reserved.

5. In Step 1, press the Browse button to select an existing package for instances or create a new package (e.g. Control_Instances) as shown below. If selecting an existing package, press OK. If creating a new package, press Enter and then OK.

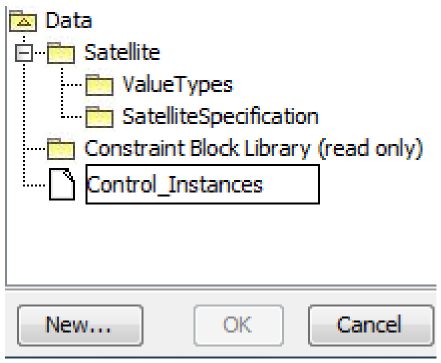
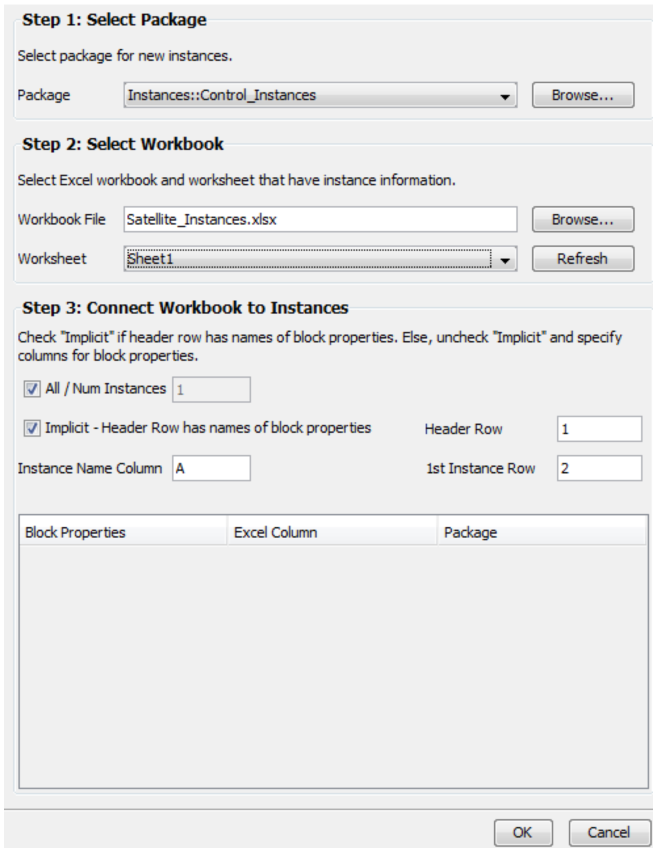


Figure 43: Creating a new package to store instances

6. In Step 2, select the workbook and worksheet containing the instance table (Figure 39). An example screenshot of the instance creation utility at the end of this step is shown below.



Step 1: Select Package

Select package for new instances.

Package: Instances::Control_Instances — Browse...

Step 2: Select Workbook

Select Excel workbook and worksheet that have instance information.

Workbook File: Satellite_Instances.xlsx — Browse...

Worksheet: Sheet1 — Refresh

Step 3: Connect Workbook to Instances

Check "Implicit" if header row has names of block properties. Else, uncheck "Implicit" and specify columns for block properties.

[✓] All / Num Instances: 1

[✓] Implicit - Header Row has names of block properties — Header Row: 1

Instance Name Column: A — 1st Instance Row: 2

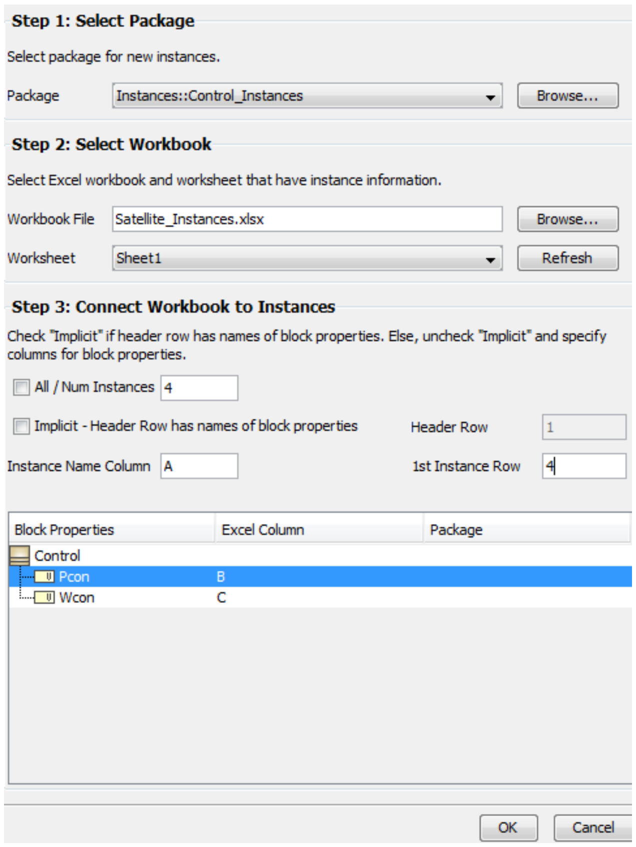
Block Properties	Excel Column	Package

Figure 44: ParaMagic instance creation utility after specifying package and workbook/worksheet information

7. In Step3, you will connect the SysML model to the Excel spreadsheet, i.e. specify the mapping between the properties of the SysML block that you want to instantiate and the columns in the Excel spreadsheet. There are 2 ways to specify this mapping:

a. *Implicit* – In this mode, the header row of the Excel table has the names of block properties. For the table shown in Figure 39, this is not the case. The property names Power and Weight in the header row (Row 4) are not the same as names of value properties of the Control block (PCon and WCon). Check *Implicit* if this is the case for your models, as shown in Figure 43. If the *Implicit* option is selected, specify the header row number (e.g. 4).

b. *Explicit* – In this mode, users have to explicitly specify what columns in the Excel table correspond to the block properties. For e.g. specify column B and C for PCon and WCon as shown below. Specify the columns that contain the instance names (e.g. A) and the first instance row (e.g. 4).


Step 1: Select Package

Select package for new instances.

Package: Instances::Control_Instances

Step 2: Select Workbook

Select Excel workbook and worksheet that have instance information.

Workbook File: Satellite_Instances.xlsx

Worksheet: Sheet1

Step 3: Connect Workbook to Instances

Check "Implicit" if header row has names of block properties. Else, uncheck "Implicit" and specify columns for block properties.

[ ] All / Num Instances: 4

[ ] Implicit - Header Row has names of block properties — Header Row: 1

Instance Name Column: A — 1st Instance Row: 4

Block Properties	Excel Column	Package
Control		
Pcon	B	
Wcon	C	

Figure 45: ParaMagic instance creation utility after specifying explicit mapping between Excel columns and value properties

8. Next, select "All" if you would like to generate block instances for all the rows in the Excel table (until ParaMagic finds an empty row), or specify the number of instances (e.g. 4) as shown in Figure 42 above.

9. Press OK.

10. You should now see instances generated in the SysML model, as shown below in

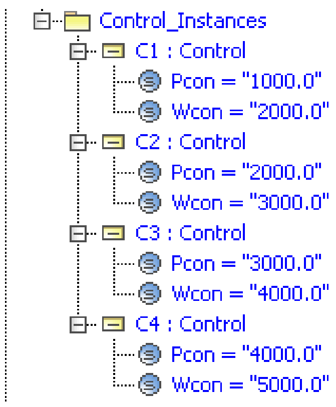
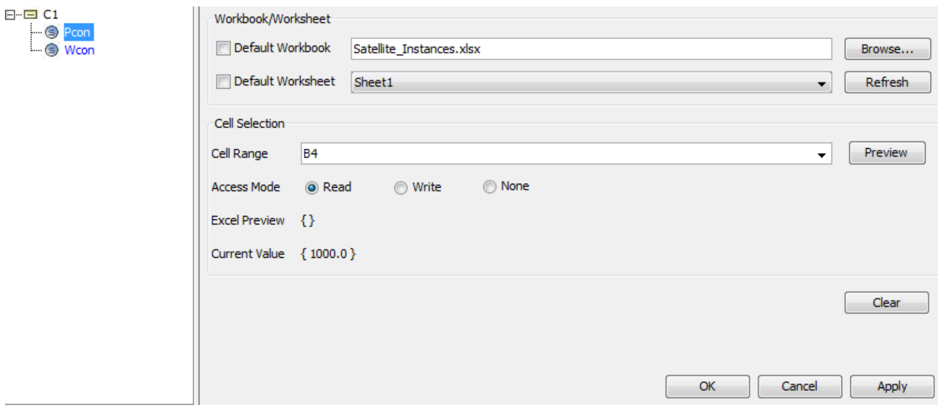

Control_Instances
C1 : Control
Pcon = "1000.0"
Wcon = "2000.0"
C2 : Control
Pcon = "2000.0"
Wcon = "3000.0"
C3 : Control
Pcon = "3000.0"
Wcon = "4000.0"
C4 : Control
Pcon = "4000.0"
Wcon = "5000.0"

Figure 46: SysML instances created by ParaMagic from Excel table

11. When generating instances from Excel spreadsheets, ParaMagic also establishes a connection between instance slots and the cells in Excel tables. This allows users to automatically read/write values from/to Excel spreadsheets to/from SysML instances. For e.g. right click the instance C1 in the MagicDraw containment tree (shown above) and select **ParaMagic > Excel > Setup**. You can see the connection between the slots (PCon and WCon) of C1 and the corresponding cells in Excel tables. If you update values in the Excel table and invoke **ParaMagic > Excel > Read** on the individual SysML instances or the owning package (e.g. Control_Instances), ParaMagic will update the values in the SysML instance model.



Default Workbook: Satellite_Instances.xlsx
Default Worksheet: Sheet1
Cell Range: B4
Access Mode: [✓ Read] Write None
Excel Preview: { }
Current Value: { 1000.0 }

Figure 47: Instances generated by ParaMagic are connected to Excel tables for value sync.

12. If you attempt to generate instances from Excel tables in a package that already contains instances with the same names as you are attempting to generate, ParaMagic will offer to overwrite the instances in the package.

8.2.1.2 Creating instances of a complex block

1. Suppose you have a complex block with several part/reference/shared properties as shown below for the SatelliteSystem block. In this example, the SatelliteSystem block has four part properties.

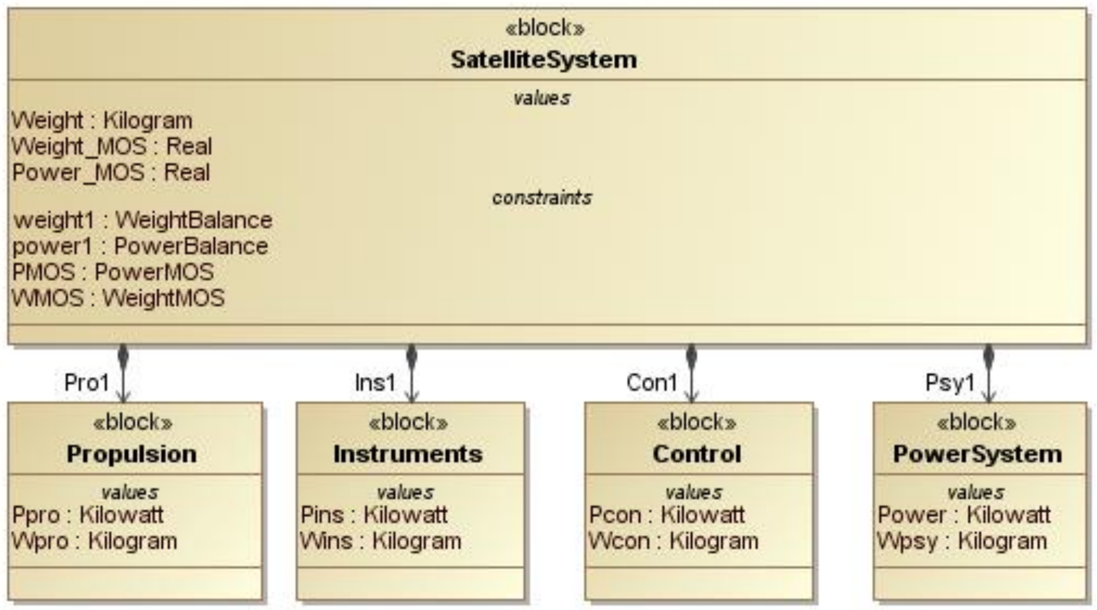
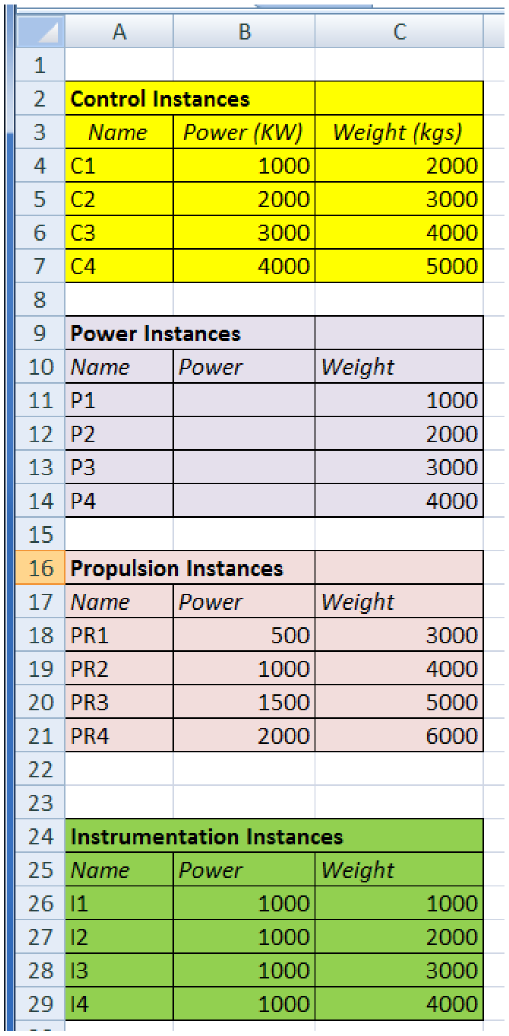


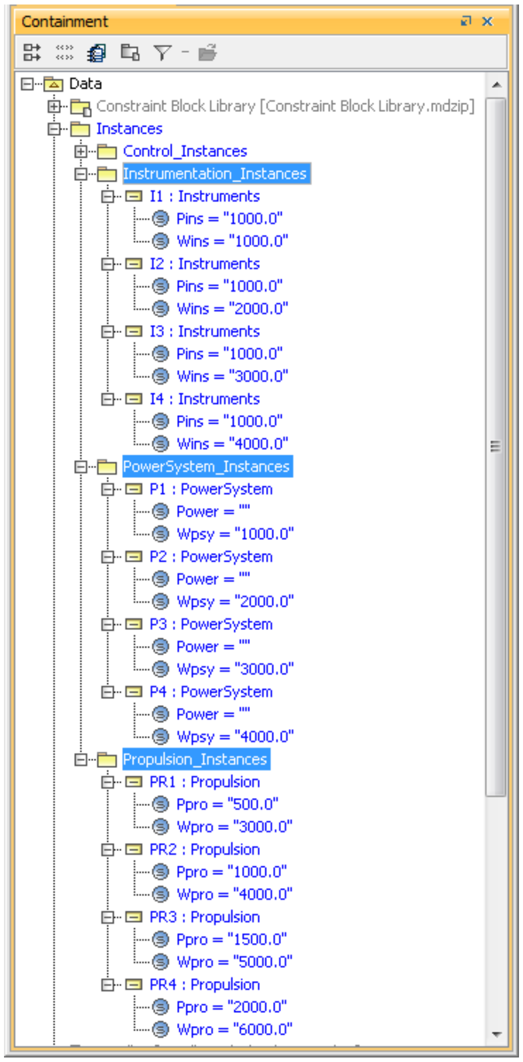
Figure 48: Example of a complex block (SatelliteSystem)

2. First, generate instances for each of the primitive blocks from Excel tables, such as instances of Control, PowerSystem, Propulsion, and Instruments blocks from the tables shown in Figure 49 below. Use steps 5-9 of the process outlined in section 8.2.1.1 above. Example SysML instances generated for the table in Figure 49 are shown in Figure 64 below.



	A	B	C
1			
2	**Control Instances**		
3	*Name*	*Power (KW)*	*Weight (kgs)*
4	C1	1000	2000
5	C2	2000	3000
6	C3	3000	4000
7	C4	4000	5000
8			
9	**Power Instances**		
10	*Name*	*Power*	*Weight*
11	P1		1000
12	P2		2000
13	P3		3000
14	P4		4000
15			
16	**Propulsion Instances**		
17	*Name*	*Power*	*Weight*
18	PR1	500	3000
19	PR2	1000	4000
20	PR3	1500	5000
21	PR4	2000	6000
22			
23			
24	**Instrumentation Instances**		
25	*Name*	*Power*	*Weight*
26	I1	1000	1000
27	I2	1000	2000
28	I3	1000	3000
29	I4	1000	4000

Figure 49: Excel tables for Control, PowerSystem, Propulsion, and Instrumentation instances

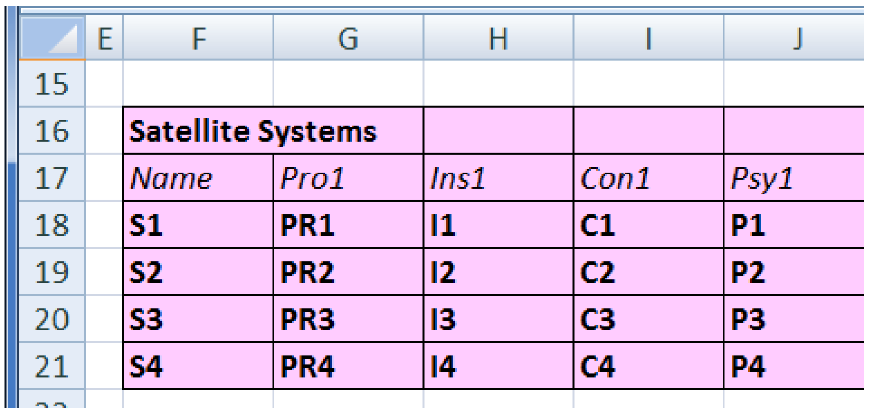


Containment

```
Data
  Constraint Block Library [Constraint Block Library.mdzip]
  Instances
    Control_Instances
    Instrumentation_Instances
      I1 : Instruments
        Pins = "1000.0"
        Wins = "1000.0"
      I2 : Instruments
        Pins = "1000.0"
        Wins = "2000.0"
      I3 : Instruments
        Pins = "1000.0"
        Wins = "3000.0"
      I4 : Instruments
        Pins = "1000.0"
        Wins = "4000.0"
    PowerSystem_Instances
      P1 : PowerSystem
        Power = ""
        Wpsy = "1000.0"
      P2 : PowerSystem
        Power = ""
        Wpsy = "2000.0"
      P3 : PowerSystem
        Power = ""
        Wpsy = "3000.0"
      P4 : PowerSystem
        Power = ""
        Wpsy = "4000.0"
    Propulsion_Instances
      PR1 : Propulsion
        Ppro = "500.0"
        Wpro = "3000.0"
      PR2 : Propulsion
        Ppro = "1000.0"
        Wpro = "4000.0"
      PR3 : Propulsion
        Ppro = "1500.0"
        Wpro = "5000.0"
      PR4 : Propulsion
        Ppro = "2000.0"
        Wpro = "6000.0"
```

Figure 50: Instances generated from Excel tables for Control, PowerSystem, Propulsion, and Instrumentation

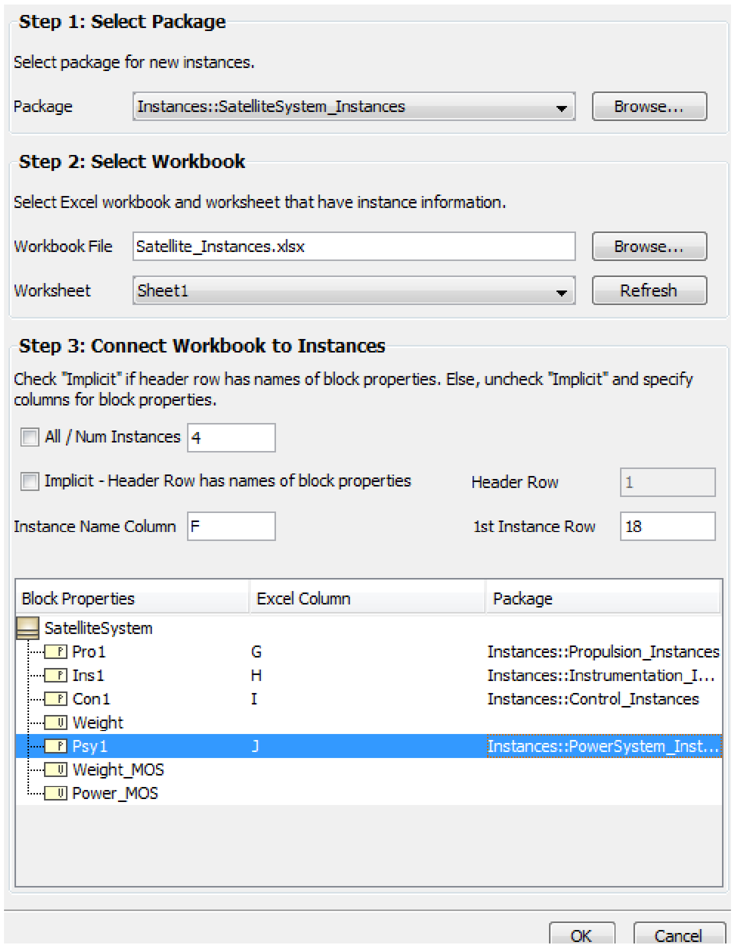
3. Now suppose you have an Excel table (as below) that shows different instances of the SatelliteSystem and the corresponding instances of the Control, PowerSystem, Propulsion, and Instruments block.



	E	F	G	H	I	J
15						
16		**Satellite Systems**				
17		*Name*	*Pro1*	*Ins1*	*Con1*	*Psy1*
18		**S1**	**PR1**	**I1**	**C1**	**P1**
19		**S2**	**PR2**	**I2**	**C2**	**P2**
20		**S3**	**PR3**	**I3**	**C3**	**P3**
21		**S4**	**PR4**	**I4**	**C4**	**P4**

Figure 51: Excel table showing instances of a complex block (e.g. SatelliteSystem)

4. To generate instances of the SatelliteSystem block, right click on the block in the MagicDraw Containment tree, and select **ParaMagic > Excel > Create Instances from Excel**. You will see the ParaMagic instance creation utility, as shown below.



Step 1: Select Package

Select package for new instances.

Package: Instances::SatelliteSystem_Instances

Step 2: Select Workbook

Select Excel workbook and worksheet that have instance information.

Workbook File: Satellite_Instances.xlsx

Worksheet: Sheet1

Step 3: Connect Workbook to Instances

Check "Implicit" if header row has names of block properties. Else, uncheck "Implicit" and specify columns for block properties.

All / Num Instances: 4

Implicit - Header Row has names of block properties

Header Row: 1

Instance Name Column: F

1st Instance Row: 18

Block Properties	Excel Column	Package
SatelliteSystem		
Pro1	G	Instances::Propulsion_Instances
Ins1	H	Instances::Instrumentation_I...
Con1	I	Instances::Control_Instances
Weight		
Psy1	J	Instances::PowerSystem_Inst...
Weight_MOS		
Power_MOS		

Figure 52: ParaMagic instance creation utility – creating instances of SatelliteSystem block

5. For generating instances of complex blocks, you must use the Explicit mode. Set Num Instances (e.g. 4), Instance Name Column (e.g. F) and 1st Instance Row (e.g. 18) based on the Excel table (e.g. Figure 65).

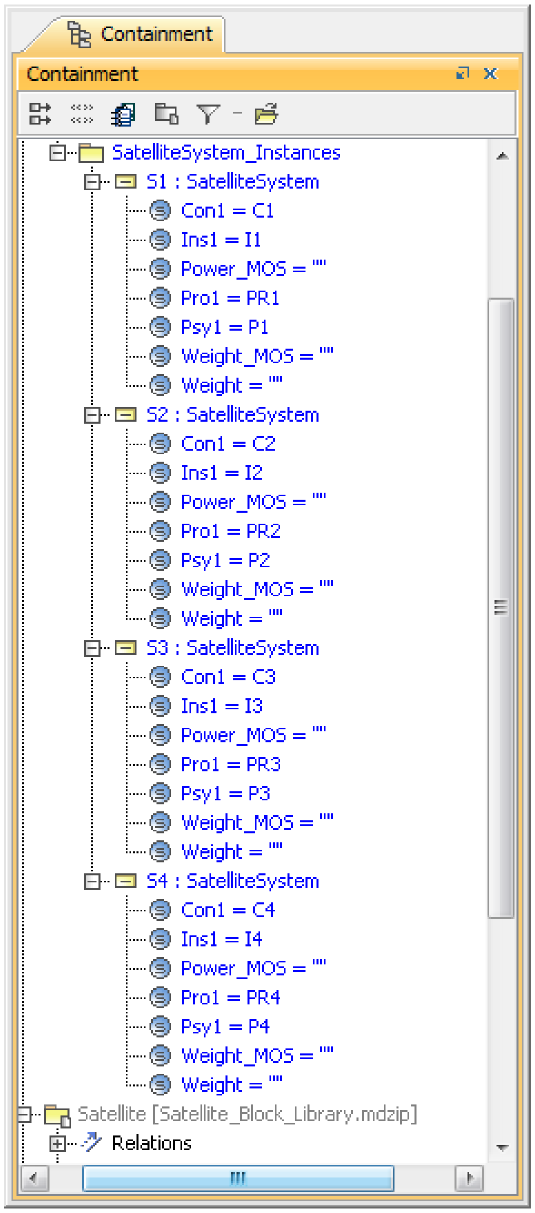
6. Specify the Excel columns corresponding to the part properties of SatelliteSystem block, such as column G for Pro1 (Propulsion instances), column H for Ins1 (Instruments instances), and so on.

7. For each part property, select the package in the SysML model that contains the instances used in the Excel table. For e.g. the package Control_Instances contains instances of the Control block (C1, C2, C3, and C4). ParaMagic will search these packages to find instances based on the names used in the Excel table (e.g. PR1, I1, C1, and P1). If unsuccessful, the part properties (slots) for the SatelliteSystem instances will not be populated.

8. Press OK

Copyright © 2014 InterCAX LLC. All Rights Reserved.

9. The MagicDraw containment tree should show you all the four SatelliteSystem instances and references to corresponding instances of Control, Power, Instrumentation, and Propulsion blocks, as shown below. Note that this instance model corresponds to the Excel table shown in Figure 51 above.



```
SatelliteSystem_Instances
  S1 : SatelliteSystem
    Con1 = C1
    Ins1 = I1
    Power_MOS = ""
    Pro1 = PR1
    Psy1 = P1
    Weight_MOS = ""
    Weight = ""
  S2 : SatelliteSystem
    Con1 = C2
    Ins1 = I2
    Power_MOS = ""
    Pro1 = PR2
    Psy1 = P2
    Weight_MOS = ""
    Weight = ""
  S3 : SatelliteSystem
    Con1 = C3
    Ins1 = I3
    Power_MOS = ""
    Pro1 = PR3
    Psy1 = P3
    Weight_MOS = ""
    Weight = ""
  S4 : SatelliteSystem
    Con1 = C4
    Ins1 = I4
    Power_MOS = ""
    Pro1 = PR4
    Psy1 = P4
    Weight_MOS = ""
    Weight = ""
Satellite [Satellite_Block_Library.mdzip]
  Relations
```

Figure 53: MagicDraw Containment tree after generating instances of SatelliteSystem

10. Similar to generating instances for primitive blocks, ParaMagic also creates Excel connections between slots (corresponding to value properties) and Excel tables when generating instances for complex blocks. Invoke ParaMagic > Excel > Read from Excel or Write to Excel to read/write values from/to Excel tables to/from SysML model.

11. **Updating instance structure in Excel tables and re-generating**: If you change relationships between instances in the Excel table, such as changing the Instruments and Control block instances associated with SatelliteSystem S1 to I2 and C2 (originally I1 and C1) respectively, and try to re-generate SysML instances from the updated table in the same package that contains instances S1-S4, ParaMagic will offer to update the SysML instances. For the example shown below, ParaMagic will update the values of slots Ins1 and Con1 of the instance S1 to I2 and C2 respectively.

Satellite Systems				
Name	*Pro1*	*Ins1*	*Con1*	*Psy1*
S1	**PR1**	**I2**	**C2**	**P1**
S2	**PR2**	**I2**	**C2**	**P2**
S3	**PR3**	**I3**	**C3**	**P3**
S4	**PR4**	**I4**	**C4**	**P4**

Figure 54: Updating instance structure in Excel table

8.2.2 Update SysML Instance Model Values from Excel Spreadsheets

If you generate instances from Excel tables, as demonstrated in section 8.2.1, ParaMagic® sets up connections between the generated instances and the Excel tables for automated reading/writing of values between SysML instance slots and Excel cells. However, if you have manually created block instances in a SysML model and intend to connect them to Excel spreadsheet(s) to read/write values, follow the steps described below.

For reference, the Orbital tutorial provided with ParaMagic™ presents a usage of this feature. Refer to the tutorial document and the Orbital model at locations shown below.

- ParaMagic® tutorial document : **<MD_Root>\samples\ParaMagic\Tutorials\Tutorials.pdf**
- Orbital model : **<MD_Root>\samples\ParaMagic\Tutorials\Orbital**

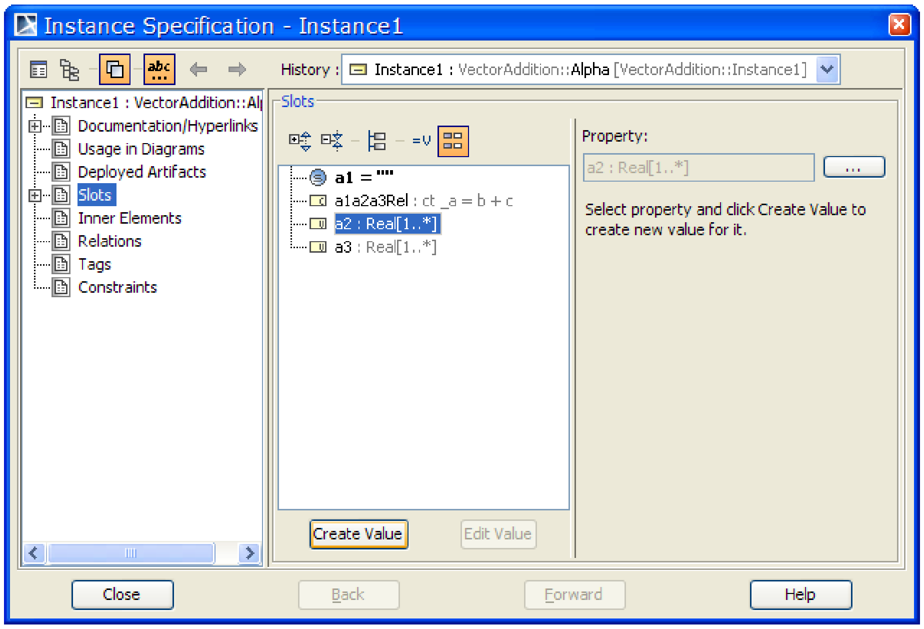
8.2.2.1 Operation

Follow the steps below to connect slots of manually created instances to Excel spreadsheets.

Step 1. *Initialize slot*

Each slot that needs to interact (read/write) with Excel spreadsheets must be initialized. Initialization ensures that the slots are visible in the MagicDraw containment tree from where ParaMagic® operations can be invoked. For initializing a slot:

a) Double click an instance in the MD containment tree. A specification window opens as shown below in Figure 55.
b) Select a slot that needs to interact with Excel spreadsheets and click on **Create Value** button. For example, slot a2 is selected in Figure 55.
c) After initialization, a slot will appear with one empty value, as shown for slot a1 in Figure 55. Initialized slots will also be visible in the containment tree under the parent instance, as shown for slots **a1**, **a2**, and **a3** of **Instance1** in Figure 56.



Select property and click Create Value to create new value for it.

Figure 55: Initialize slots

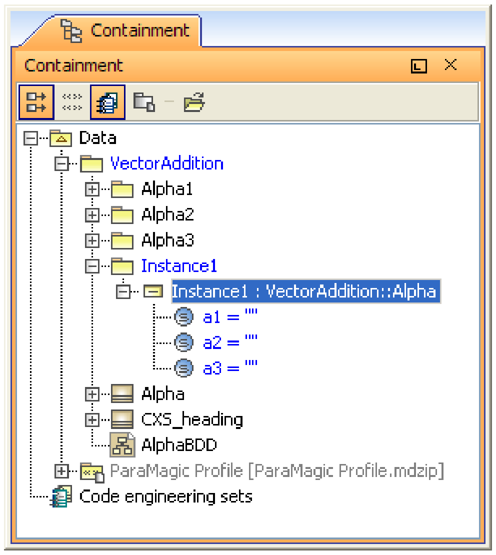


Figure 56: MagicDraw Containment tree after initialization of slots.

For multi-valued slots (e.g. slots that store an array of numbers), the initialization process is the same as above. For initializing multi-valued slots that are not inputs (givens) for ParaMagic™ simulations, see step 9 below.

Step 2. *Setup Excel slots to interact with Excel worksheet*

To setup slots to interact with Excel, right click on the slot (or its parent instance) in the MD containment tree and select **ParaMagic > Excel > Setup**, as shown in Figure 57 below. The ParaMagic Excel Setup utility appears, as shown in Figure 58 below.

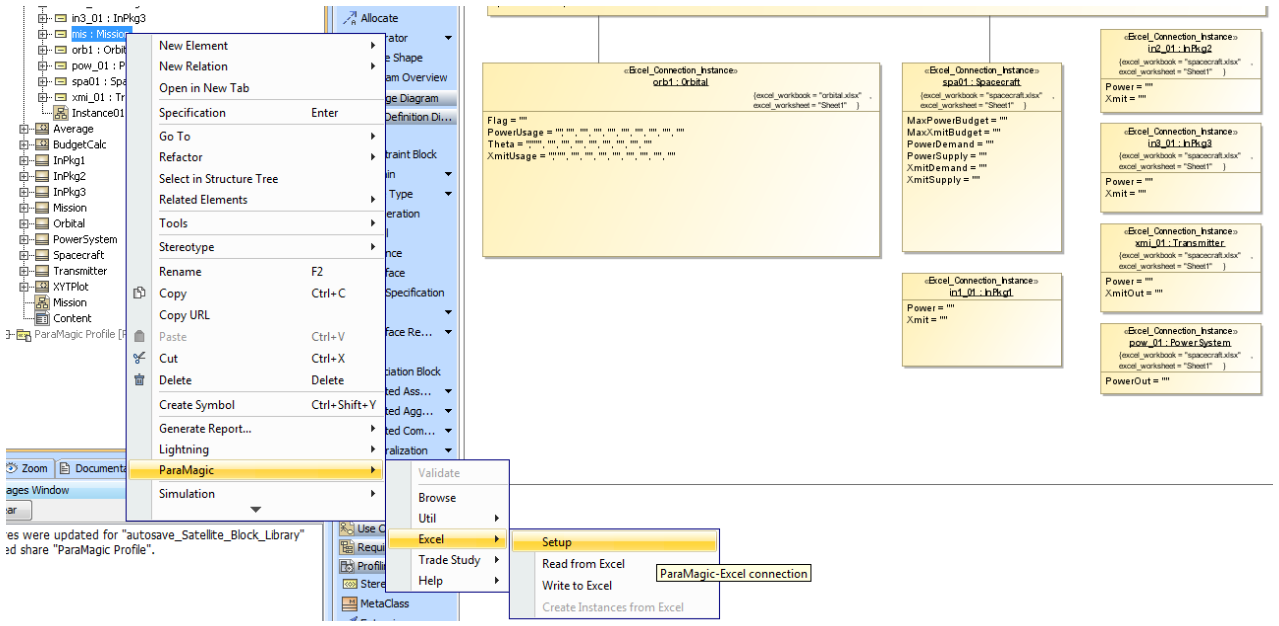
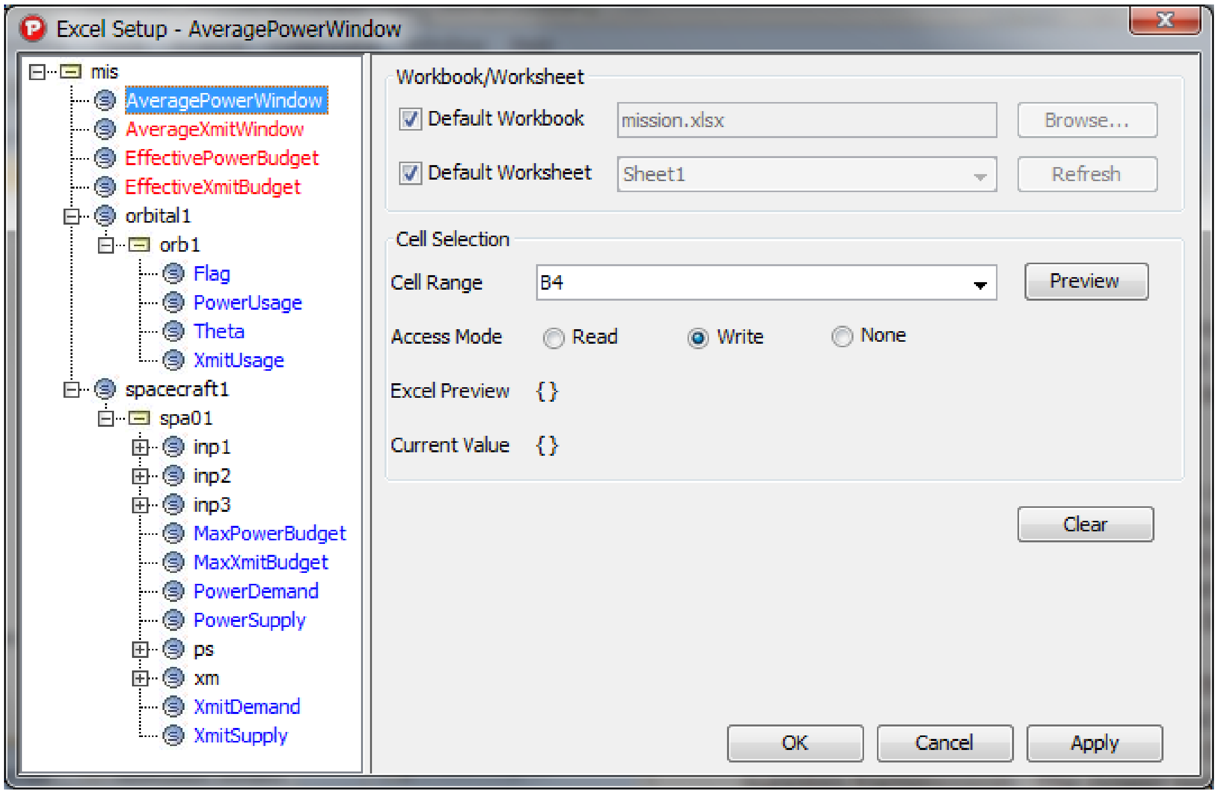


Figure 57: Launch Excel Setup utility for an instance/slot



Workbook/Worksheet

[✓] Default Workbook: mission.xlsx

[✓] Default Worksheet: Sheet1

Cell Selection

Cell Range: B4

Access Mode: Read [✓ Write] None

Excel Preview: {}

Current Value: {}

Figure 58: ParaMagic Excel Setup utility

The ParaMagic® Excel Setup utility can be invoked for a slot, instance, or a package for which a default instance[19] has been selected. When invoked, users can see the entire instance model in the

[19] For details, see the command Util > Set default instance in section 7.1

Copyright © 2014 InterCAX LLC. All Rights Reserved.

context of the selected instance. Then, users can click on specific slots and link them to Excel spreadsheets.
If Excel connection-related information is already populated for the slot, corresponding values will be shown in the setup window as shown in Figure 58 above.

Step 3. *Specify Excel connection for slots*

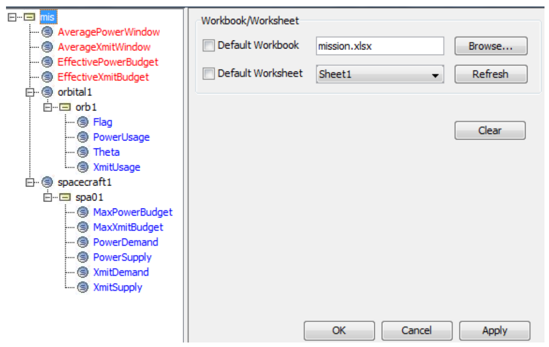
Specify Excel connection information for all slots that need to be linked to Excel spreadsheets. To do this, repeat steps a-f below for all such slots.

a) Select the slot in the instance model shown in the setup utility (left pane). For example, the slot **AveragePowerWindow** is shown selected in Figure 58.

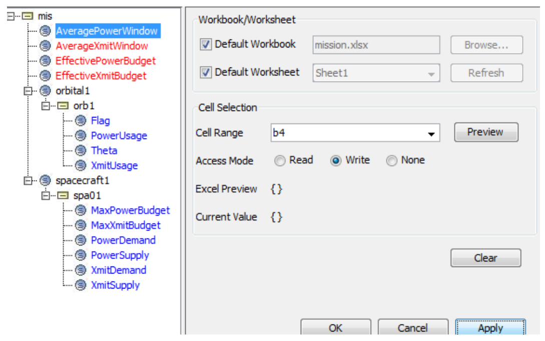
b) Specify the location of the Excel workbook associated with the slot in the *Workbook File* field. If the workbook file is located in the same folder as the MagicDraw file (mdzip), only the name of the workbook file needs to be specified (with the extension **.xls** or **.xlsx**). Alternatively, click the **Browse** button to select the workbook file. Note that storing Excel workbooks with the MagicDraw file enhances the portability of the model. The MagicDraw model file and the workbook files can be distributed together without system-specific folder location settings.

c) Select the worksheet associated with the slot. If you selected the workbook file using the **Browse** button, the list of spreadsheets in that file is automatically available for selection. However, if you manually entered the workbook file, click on the **Refresh** button to populate the list of spreadsheets in the workbook file. Note that if the workbook file is not found, the list of available worksheets will be empty.

If multiple slots owned by a single instance are linked to the same workbook and worksheet, it is preferable to specify workbook and worksheet information at the instance level and then select the **Use Default Worksheet** option for all such slots. Figure 59 below shows an example where the Excel workbook and worksheet is specified for the instance **mis**. The slot **AveragePowerWindow** (owned by this instance) uses the worksheet.



Workbook/Worksheet
Default Workbook [ ] mission.xlsx
Default Worksheet [ ] Sheet1



Workbook/Worksheet
Default Workbook [✓] mission.xlsx
Default Worksheet [✓] Sheet1
Cell Selection
Cell Range b4
Access Mode Read [✓ Write] None
Excel Preview {}
Current Value {}

Excel worksheet specified for the instance Slot uses the default worksheet

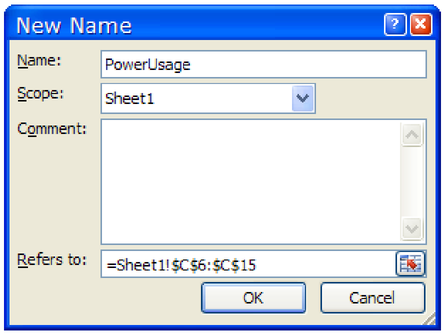
Figure 59: Specifying workbook and worksheet at the instance level and using it for slots

Note that ParaMagic® allows you to propagate default workbook and worksheet settings through the instance hierarchy. For e.g. workbook and worksheets specified at the top-level instance (e.g. mis) can be used as defaults for all lower level slots and instances (at any depth in the model).

d) Specify the cell range (in the selected worksheet) associated with the slot. Cell range can be specified using cell name or address.

If the cell range is specified using a name, the following rule applies:

i) The scope of the name should be the worksheet and not the workbook. For example, as shown in Figure 60 below, cell range **C6:C15** in **Sheet1** is named as **PowerUsage**. The scope of this name is the worksheet **Sheet1**.



New Name

Name: PowerUsage

Scope: Sheet1

Comment:

Refers to: =Sheet1!$C$6:$C$15

Figure 60: Specify cell range using name

If the cell range is specified using cell address, the following rules apply:

i) Cell range must be specified using the following Excel syntax:
   (1) for multi-valued slots: **<first cell address>:<last cell address>**. For example, if a multi-valued slot is to be associated with cells from A2 to A10, cell range should be specified as **A2:A10**.
   (2) for single-valued slot: **<cell address>**. For example, if a single-valued slot is associated with cell A2, cell range should be specified as **A2**.

ii) Cell ranges must be specified without the worksheet reference. For example, cell range A2 to A10 in MySheet worksheet should be specified as **A2:A10** and not as **MySheet1!A2:A10**.

iii) A cell range should not have special characters (including whitespaces) that Excel cannot recognize. The following are examples of syntactically incorrect cell ranges: **A2 : A10**, **A 2 : A 10**, **A$%2:A*9**

iv) For associating a SysML slot to an Excel spreadsheet using ParaMagic®, cell ranges must be contiguous and correspond to a single column/row. For example, **A2:A10** and **A2:F2** are contiguous and correspond to single column and row cell ranges respectively; **A2:B10** is a contiguous but not a single column/row cell range; and **A2:A10,C2:C10** is a valid but non-contiguous cell range.

v) ParaMagic® ignores the order in which the first and last cells are specified in the cell range field. For example, **A10:A2** is treated the same as **A2:A10**.

e) Specify the access mode (Read or Write).

   If the Access Mode=**Read**, ensure that the workbook file is saved before the Excel Read operation is executed. If Access Mode=**Write**, ensure that the workbook file is closed before the Excel Write operation is executed. As shown in Figure 58, slots that are setup to read from spreadsheets are shown in blue and slots that are setup to write to spreadsheets are shown in red.

f) Press the **Apply** button to save slot setup information to the MagicDraw model. Those slots for which the Excel connection settings have been changed but not saved to the MagicDraw model (**Apply** button) are shown in *italics*.

The table below summarizes the commands that can be issued from the ParaMagic® Excel Setup utility.

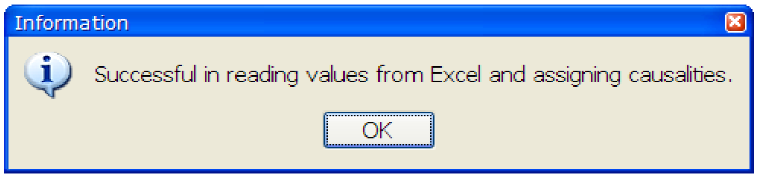
Command	Description
Browse	Opens a file browser for selecting an Excel workbook file (.xls or .xlsx)
Refresh	Creates a list of spreadsheets available in the specified workbook file
Apply	Saves the Excel connection settings to the MagicDraw model
OK	Saves the Excel connection settings to the MagicDraw model (Apply) and closes the setup utility
Clear	Removes the Excel connection settings for the subject slot
Cancel	Closes the ParaMagic® Excel Setup utility without saving the Excel connection settings

Note: Since the workbook and worksheet information is specified for a particular slot, different slots in an instance may be connected to different workbooks/worksheets and setup to Read/Write information from/to Excel spreadsheets.

Step 4. *Execute Excel Read / Write operation for the setup slot*

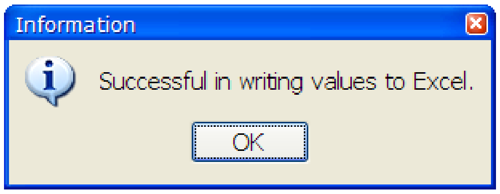
Once a slot is setup, Excel Read or Write operations can be executed by right clicking the slot and selecting **ParaMagic > Excel > Read from Excel** or **Write to Excel** menus. When the Read or Write operation is executed successfully, corresponding information messages pop-up as shown in Figure 61.

During the Excel Read operation, the causality of all slot values read from Excel is automatically changed to "**given**" (from "**undefined**") in accordance with Table 1. Once a slot has been connected to an Excel worksheet, the Excel Read/Write operations can be invoked whenever new values are to be read from (or written to) the Excel worksheet.



Successful in reading values from Excel and assigning causalities.

Message shown for successfully reading slot values from Excel



Successful in writing values to Excel.

Message shown for successfully writing slot values to Excel

Figure 61: Information messages for successful execution of ParaMagic® Excel Read/Write operations

Steps 1-4 above demonstrate the basic use of ParaMagic™ to (1) connect SysML instance slots to Excel spreadsheets, and (2) read/write values from/to spreadsheets to/from instance slots. Additional steps for more efficient operations are described below.

Step 5. *Execute Excel Read/Write operation for an instance or an instance package*

Excel Read/Write operations can be invoked for a slot (as demonstrated above), an instance, or a package containing instances. To invoke Excel Read/Write operation on an instance (or instance package), right click the instance (or instance package) and select **ParaMagic→Excel→Read from Excel** or **Write to Excel** menu.

If the Excel Read operation is invoked on an instance, ParaMagic® will execute the Excel Read operation for all slots of that instance that are setup to read from Excel (Access Mode=**Read**). Similarly, if the Excel Write operation is invoked on an instance, ParaMagic® will execute the Excel Write operation for all slots of that instance that are setup to write to Excel (Access Mode= **Write**).

If the Excel Read/Write operation is invoked on a package, ParaMagic® will perform the Excel Read/Write operation for all instances in the instance package.

Step 6. *Initializing slots with empty values before solving with ParaMagic.*

If a slot value is intended to be a target or an undefined variable for ParaMagic® solution, users are still required to initialize the slot (as shown in step 1 above). If the slot value is an array, then **n** values of that slot must be initialized (where **n** is the array size). This can be a cumbersome process. Using ParaMagic®, users can initialize a slot with **n** empty values by setting them up to read values from **n** empty cells in an Excel spreadsheet and then executing the Excel Read operation. If the slot values do not have any pre-assigned causality, the Excel Read operation will set their causality to "undefined". For those empty values that are the targets, users can change the causality from “undefined” to “target” in the ParaMagic™ browser.

The table below summarizes the ParaMagic® commands for Excel setup/read/write, the model elements for which these commands may be issued, their behavior, and the response message on successful execution of commands. These commands are available under **ParaMagic > Excel** menu.

Command	Arguments	Description	Messages
Setup	Slot	Opens the Excel Setup utility for the instance owning the slot. The subject slot is selected.	After the **Apply** (or **OK**) button is pressed, the setup values are saved to the model. No messages pop-up after the save operation is completed.
	Instance	Opens the Excel Setup utility for the instance. The subject instance is highlighted in the instance model.	
	Package	Opens the Excel Setup utility for the default instance in this package. To set a default instance, see the command Util > Set default instance in section 7.1	
Read	Slot	Reads values from an Excel spreadsheet and populates slot values if slot access mode=**Read.** The causality is set to "given" for all slot values read from Excel	• If slot values are read successfully from Excel and the causality assignment is successful, the response message states "Successful in reading values from Excel and assigning

			default causalities." • If slots values are successfully read from Excel but the causality assignment is unsuccessful, the response message states “Successful in reading values from Excel but unsuccessful in assigning causalities." • If slot values are not read from Excel, response messages (indicating the problems) are shown in the MagicDraw Message window.
	Instance	Executes the Excel Read operation for all slots in the instance.	A successful response message is show only when the Excel read and causality assignment operation is successful for all slots (with read access mode) in the instance. Else, no response message is shown.
	Package	Executes the Excel Read operation for all slots of all instances in the package.	A successful response message is shown only when the Excel read and causality assignment operation is successful for all slots of all instances in the instance package. Else, no response message is shown.
Write	Slot	Writes slot values to an Excel spreadsheet if slot access mode = **Write**.	• If slots values are successfully written to Excel, the response message states "Successful in writing values to Excel". • If slot values are not written to Excel, response messages indicate the problem.
	Instance	Executes the Excel Write operation for all slots in the instance.	A successful response message is shown if the Excel Write operation is successful for all slots of the instance.
	Package	Executes the Excel Write operation for all slots of all instances in the package.	A successful response message is shown only if the Excel Write operation is successful for all slots of all instances in the instance package.
Create Instances from Excel	Block	Generates instances of the block from data in Excel tables	

Copyright © 2014 InterCAX LLC. All Rights Reserved.

8.2.2.2 Features and Specific Behavior

1) Both versions of MS Excel files are supported—Excel 97-2003 files (**.xls** extension) and Excel 2007 files (**.xlsx** extension).

2) Both numeric and text (string) values can be read from Excel or written to Excel. If the block value property corresponding to a slot is typed by:
   a) value type **Real** or its subtype, ParaMagic® will read/write only numeric values for that slot.
   b) value type **String** or its subtype, ParaMagic® will read/write values for that slot as strings.
   c) any other value type, ParaMagic® will behave the same as (b) above.

3) ParaMagic® operations are one-time read/write operations and not a live connection. If values in Excel workbooks (associated with slots in MagicDraw) are updated, the Read operation must be re-invoked on all slots to read updated values after the Excel spreadsheet has been saved. Similarly, if slot values in SysML instance model (associated with values in Excel workbooks) are updated, the Write operation must be re-invoked to write updated values from SysML model to Excel spreadsheets after the workbooks have been closed.

4) Empty cells in an Excel spreadsheet are read as empty slot values in the Excel Read operation. Similarly empty slot values are written as empty cells in Excel spreadsheet in the Excel Write operation. Note that cells that look empty (no contents) may be null (esp. if they have not stored a value previously). In an Excel read operation, null cells are not read as empty slot values.

5) If a slot has "**n**" values and it is associated with "**k**" cells in a spreadsheet then the behavior of Excel Read and Write actions invoked on the slot are specified in the table below.

Excel Read operation	
n>=k	**n<k**
• Only the first **k** values of the slot (has **n** values) are updated with the values in **k** cells in the Excel spreadsheet. • The remaining values of the slot—(**k+1**)th value to the **n**th value—are deleted. • After a read operation, causality is assigned based on Table 1 – if slot entry has a value, then set to given or keep as target. If slot entry is null, then assign/keep as undefined.	• The **n** values in the slot are updated with the values in first **n** (out of **k**) cells in the Excel spreadsheet. • New slot values are created for values of (**n+1**)th to the **k**th cell in the Excel spreadsheet. • Causality is assigned based on Table 1 – if slot entry has a value, then set to given or keep as target. If slot entry is null, then assign/keep as undefined.

Excel Write operation	
n>k	**n<k**
Excel write operation will not work if **n** is not equal to **k**. This is to avoid overwriting existing cells in spreadsheets for ambiguous cases such as these.	

8.2.2.3 Limitations

1) The Excel workbook needs to be closed before the Excel Write operation is invoked on a slot or instance or instance package.
2) The Excel Write operation will not create a new workbook file if it does not exist.
3) ParaMagic® ignores the order in which the first and last cells are specified in the cell range field. For example, **A10:A2** is treated the same as **A2:A10**.

Note: Although ParaMagic® is designed to ensure that existing features in Excel workbooks, such as charts, formulas, macros, and pivot tables, are preserved when reading and writing values from/to workbooks, all of these features are not tested with ParaMagic®. As an extra caution, ParaMagic® creates a backup of Excel spreadsheets, at the same location as the original spreadsheet, before the Write operation.

8.3 ParaMagic - MATLAB Connection

The ParaMagic – MATLAB connection (PM-MC) enables users to wrap MATLAB functions and scripts as SysML constraint blocks and use them in parametric models as regular constraint properties. ParaMagic™ solves for the constraints by invoking MATLAB functions/scripts when required. MATLAB scripts are commonly used to invoke and execute Simulink models. Since PM-MC can solve for constraints that wrap MATLAB scripts, it can be used to execute Simulink models and feed the results of the execution back into SysML models.

The HomeHeating tutorial provided with ParaMagic is an example usage of the PM-MC feature. It is located under **<MD_Root>\samples\ParaMagic\Tutorials\HomeHeating.** This model has been tested to work with MATLAB 2014a.

A MATLAB script[20] is used for automatically executing a series of MATLAB commands. Users that perform computations on MATLAB command line write MATLAB scripts which can be loaded in the MATLAB environment to achieve the same effect. A script has no input and output arguments. However, a script may create and access variables in MATLAB workspaces. In contrast, a MATLAB function[20] accepts input arguments and has output arguments.

MATLAB scripts and functions are written in MATLAB files—commonly known as M-files. These files also have a **.m** extension like Mathematica files. Users should be careful to distinguish a **.m** file native to Mathematica versus a **.m** file native to MATLAB. A MATLAB M-file containing a script is known as a script M-file, and a MATLAB M-file containing a MATLAB function is known as a function M-file.

If all relations in your SysML model are MATLAB relations (i.e. they wrap function or script M-files), then ParaMagic™ does not require Mathematica to solve the model.

In the following two sections, the process of wrapping MATLAB scripts M-files and function M-files using constraint blocks is demonstrated. Once wrapped, ParaMagic can invoke MATLAB scripts/functions when the constraints need to be solved. Step 1 and 0 below are common to using script and function M-files.

Step 1. *Check MATLAB installation on your computer*

Follow the steps below to ensure that MATLAB is installed correctly on your computer.

1) Go to the command prompt. On Windows, you may do this by selecting **Run** from the **Start** menu and typing the command **cmd and pressing the OK button**.
2) Type **matlab** at the command line. This should launch MATLAB on your computer.
3) Before wrapping MATLAB function or script M-files, ensure that they are correct, i.e. they have valid MATLAB syntax and provide valid results for valid inputs. To do this, run the script M-file on

[20] MATLAB scripts and functions: http://www.mathworks.com/access/helpdesk/help/techdoc/matlab_prog/f7-38085.html

Copyright © 2014 InterCAX LLC. All Rights Reserved.

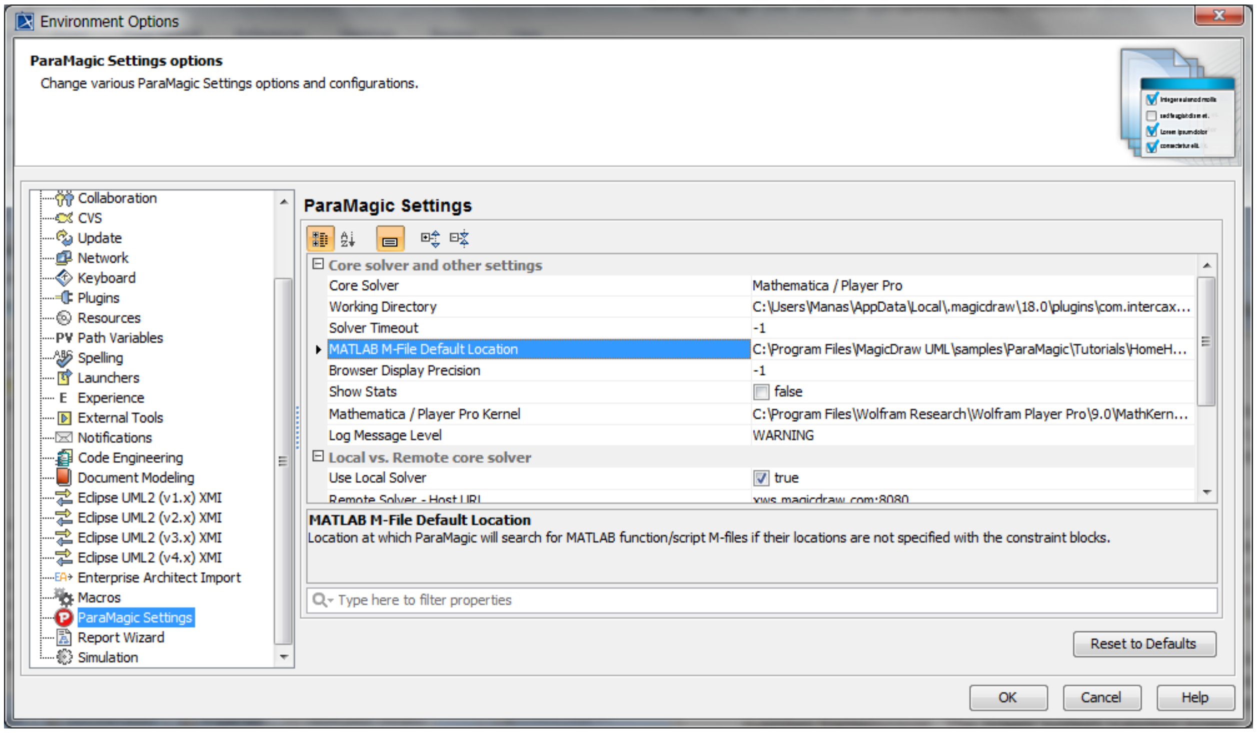
MATLAB installed on your computer, or call the function M-file from your MATLAB workspace. Once MATLAB scripts/functions have been tested to work with MATLAB, then they are ready to be used with ParaMagic™.

Step 2. *Specify default location of MATLAB function/script M-files*

It is preferred that users provide a default location (folder) of MATLAB function/script M-files. If the M-file location is not specified with the constraint block (that wraps the M-file), ParaMagic will search for the M-file in this default location. This behavior can used to your advantage if all (or most) of your M-files are at the same location (say X). If so, you do not need to specify the M-file location for each constraint block that wraps it but only specify location X in the manner described below.

To specify the parent folder location, follow the steps below:

1) Launch ParaMagic® Settings window in MagicDraw (**Options > Environment > ParaMagic Settings**), as shown in Figure 62 below.
2) Specify location of the folder as the value of the variable **MATLAB M-File Default Location**, as shown in Figure 62 below.



Core solver and other settings	
Core Solver	Mathematica / Player Pro
Working Directory	C:\Users\Manas\AppData\Local\.magicdraw\18.0\plugins\com.intercax...
Solver Timeout	-1
MATLAB M-File Default Location	C:\Program Files\MagicDraw UML\samples\ParaMagic\Tutorials\HomeH...
Browser Display Precision	-1
Show Stats	☐ false
Mathematica / Player Pro Kernel	C:\Program Files\Wolfram Research\Wolfram Player Pro\9.0\MathKern...
Log Message Level	WARNING
Local vs. Remote core solver	
Use Local Solver	✓ true
Remote Solver - Host URL	xws.magicdraw.com:8080

MATLAB M-File Default Location
Location at which ParaMagic will search for MATLAB function/script M-files if their locations are not specified with the constraint blocks.

Figure 62: Specify default location of MATLAB M-files in ParaMagic Settings window

Note: The location of a MATLAB M-file can also be specified with each constraint block that wraps the M-file.

***Step 3.** Specify a timeout for expecting results from MATLAB functions and script executions*

The **ParaMagic.ini** file includes a variable that specifies the timeout interval for ParaMagic™ when waiting for MATLAB functions and scripts to finish execution. By default, this is set to 180 seconds as shown below.

com.intercax.xaitools.solver.timeout.in.seconds=180

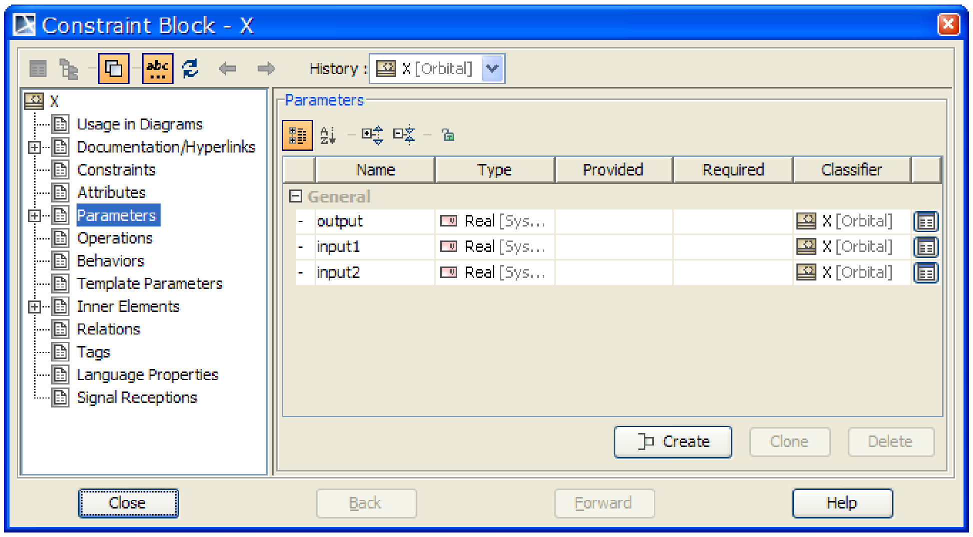
Users are recommended to specify an upper-bound value for this variable depending on the time typically taken to execute the MATLAB functions/script that they intend to use with ParaMagic™. Once the timeout is reached, ParaMagic™ stops expecting results from MATLAB but does not terminate the MATLAB execution process.

***Step 4.** Define a constraint block to "wrap" MATLAB script/function M-file*

1) Locate the **xfwExternal_Matlab_Script** or **xfwExternal_Matlab_Function** constraint block in the Constraint Block Library package loaded with the ParaMagic Profile. The former is setup to wrap script M-files and the latter is setup to wrap function M-files.

2) Copy the **xfwExternal_Matlab_Script** or **xfwExternal_Matlab_Function** constraint block to your package and rename it (say X).

3) Double click on the new constraint block X. This will open the Specification window as shown below.



Name	Type	Provided	Required	Classifier
General				
output	Real [Sys...			X [Orbital]
input1	Real [Sys...			X [Orbital]
input2	Real [Sys...			X [Orbital]

Figure 63: Constraint block specification window – constraint parameters view

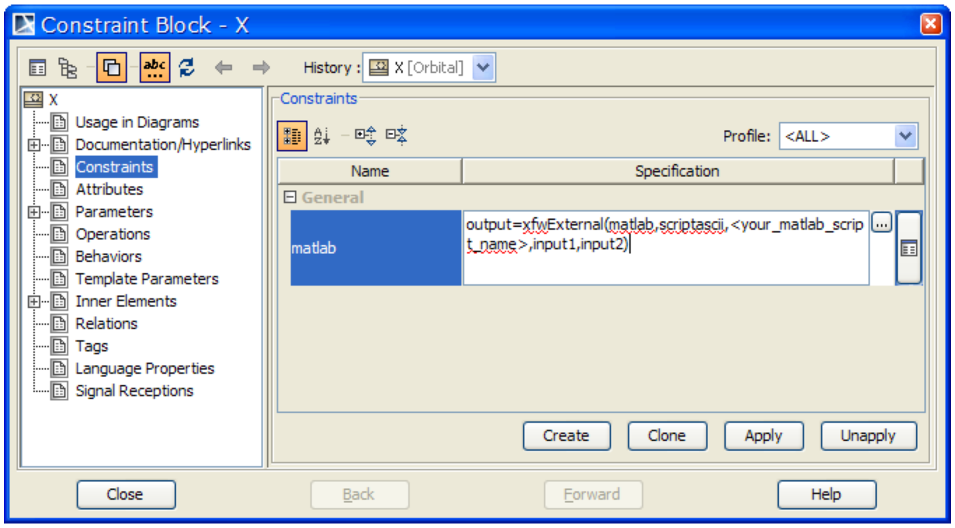
4) By default, this constraint block is setup to wrap a Matlab M-file with 2 inputs and 1 output. Add/remove input parameters depending on the number of inputs required by your M-file. Only 1 output parameter is allowed (single-valued or a single-dimensional array). For M-file functions, the input parameters and output parameters correspond to the arguments passed to and the value returned by the function. For M-file scripts, the input parameters correspond to those value properties that need to be passed to the script and the output parameters correspond to those value properties that are to be populated at the end of the script execution. See section 8.3.1 for specifics related to Matlab scripts.

5) Click on the Constraints menu in the Constraint Block specification window. By default, a constraint has been specified. Modify this constraint equation per your requirements. The general format of the constraint relation is:

<out_param> = xfwExternal(matlab, <script_or_function>, <name_of_ M-file>, <in_param_1>,...)

where:
- **<out_param>** is the name of the output parameter (result computed during execution and to be read back into SysML instance)
- **<script_or_function>** is set to **scriptascii** (for MATLAB scripts) or **function** (for MATLAB functions)
- **<name_of_M-file>** is the name of the MATLAB M-file without the extension (**.m**)
- **<in_param_1>,...** is a comma-separated list of input parameters (given values to be sent from SysML instances to MATLAB script/function)

Terms enclosed in < > are variables that can have different names, while those not enclosed in < > are keywords/constants that should not be changed.



Name	Specification
matlab	output=xfwExternal(matlab,scriptascii,<your_matlab_script_name>,input1,input2)

Figure 64: Constraint block specification window – constraints view

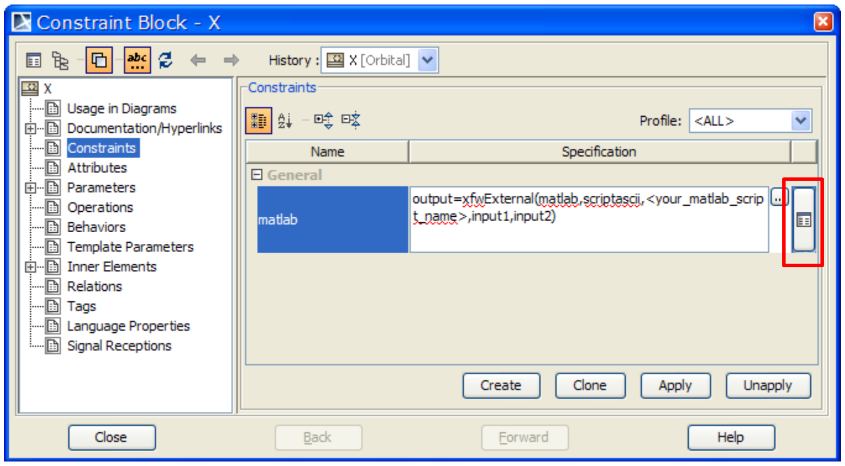
For example, Figure 69 below shows a constraint block with five parameters that wraps a **MATLAB** M-file script (**demoscriptasciisimulink.m**). These parameters correspond to the given and computed variables in the script. For example, values of **row**, **col**, **outtemp**, and **daycyc** variables are given, and the value of the variable **cost** is computed during the script execution.

In ParaMagic®, both input and output parameters of the constraint block wrapping a M-file could be single-valued or multi-valued (e.g. array).

Step 5. *Specify the location of the folder containing the M-file*

To specify the location of the folder containing the M-file (to be wrapped by the subject constraint block), follow the steps below:

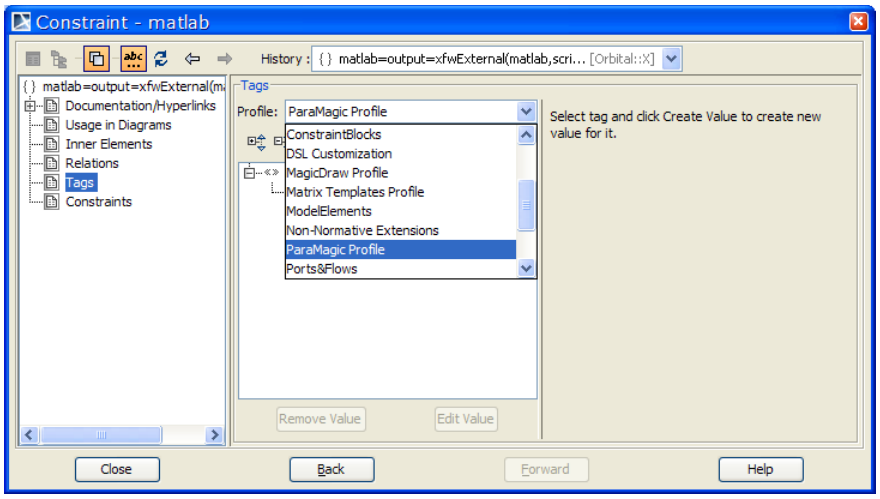
1) Click on the icon highlighted below or double click the constraint specification in the MD containment tree.



Name	Specification
General	
matlab	output=xfwExternal(matlab,scriptascii,<your_matlab_script_name>,input1,input2)

Figure 65: Invoking the constraint specification window from the constraint view

2) The action in the previous step will open the specification window for the constraint. Click on the Tags menu as shown below and select ParaMagic Profile in the Profile list. This will display all stereotypes (and their tags) that are provided by the ParaMagic Profile for constraints.



Select tag and click Create Value to create new value for it.

Figure 66: Specifying the location of M-file - selecting the ParaMagic profile

3) Select the **working_dir** tag and click on the **Create Value** button, as shown below.

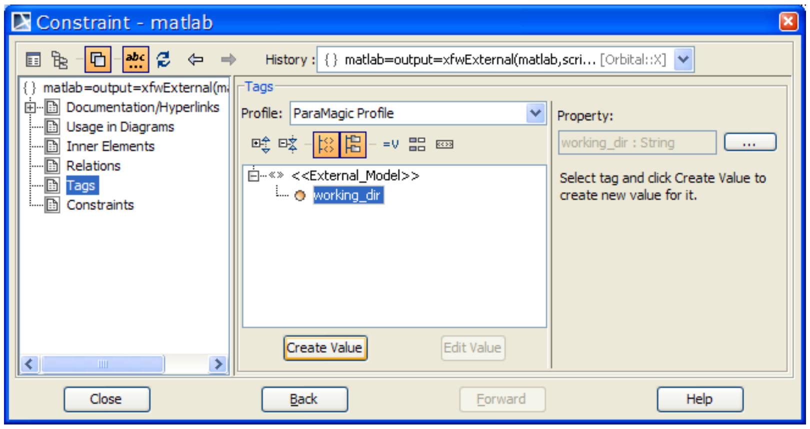


Figure 67: Specifying the location of M-file – selecting the tag to populate

4) Specify the location of the folder containing the M-file. For example, if the M-file is located under C:\Data\My_MATLAB_Files, specify the following value for the **working_dir** tag: **C:\\Data\\My_MATLAB_Files**.

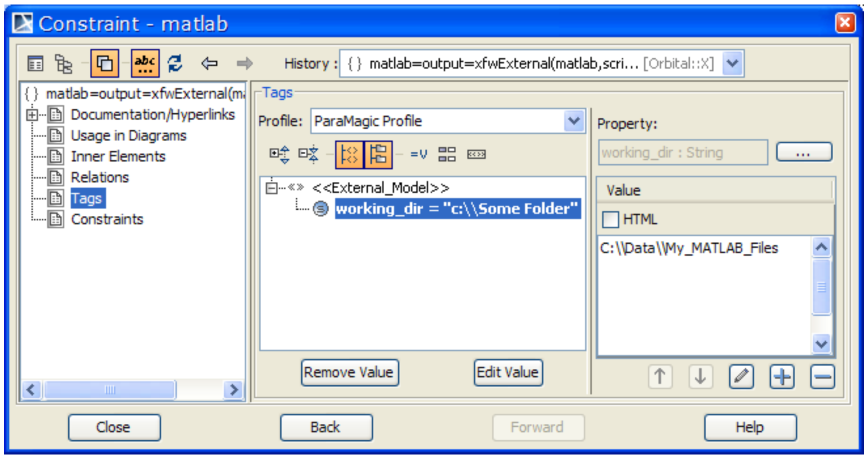


Figure 68: Specifying the location of M-file by populating the tag

Note that you only need to specify the location of the folder containing the M-file. The name of the M-file is specified in the constraint specification.

Step 6. *Define constraint properties typed by the wrapped constraint block*

After defining a constraint block that wraps MATLAB M-files, constraint properties could be defined for blocks whose values properties are to be related using the subject MATALB function/script. These constraint properties should be typed by the constraint block created in Step 4 above. When creating SysML parametric models using parametric diagrams, the constraint parameters of the constraint property should be connected to the value properties of the block. For example, as shown in Figure 71, a constraint property **SHH** (of type **SimulinkHomeHeating** constraint block) is defined for a block and its constraint parameters **outtemp**, **daycyc**, **row**, **col**, and **cost** are connected to a block's value properties **Outdoors.Temp**, **DailyCycle**, **OutputRow**, **OutputColumn**, and **DailyCost** respectively.

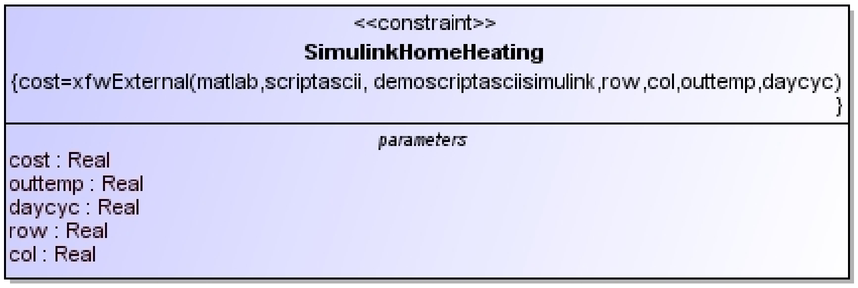
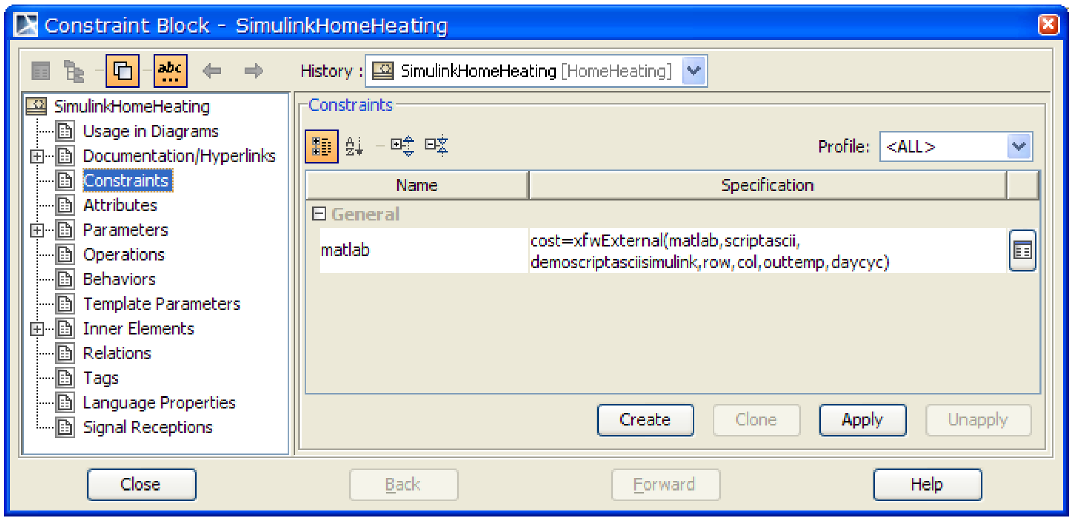


Figure 69: SysML constraint block to wrap MATLAB scripts


Constraint Block - SimulinkHomeHeating

History : SimulinkHomeHeating [HomeHeating]

Profile: <ALL>

Name	Specification
General	
matlab	cost=xfwExternal(matlab,scriptascii, demoscriptasciisimulink,row,col,outtemp,daycyc)

Figure 70: Defining constraint specification for a constraint block to wrap MATLAB script M-file

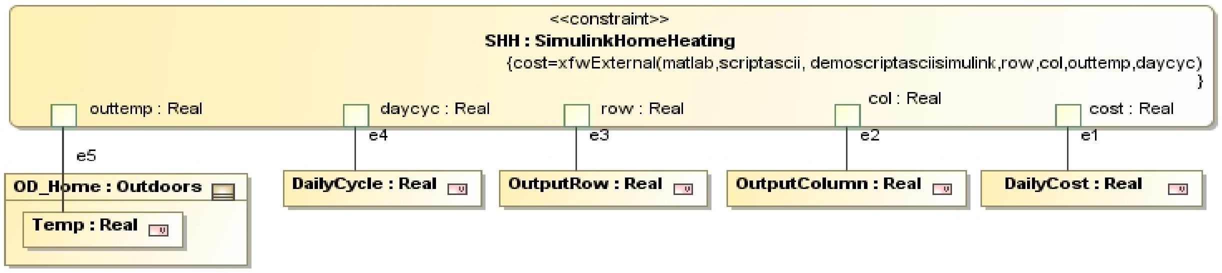


Figure 71: SysML constraint property typed by a constraint block (that wraps MATLAB scripts) and used in a SysML parametric diagram

8.3.1 Using MATLAB scripts

Unlike MATLAB functions, scripts do not have input arguments and output/return values. ParaMagic™ uses intermediate input and output files to transfer SysML instance values (givens) from MagicDraw to a MATLAB script before executing the script, and to transfer results obtained by executing the script to SysML instance values (targets). To use MATLAB scripts with ParaMagic™, follow the steps below after finishing Step 6 above.

***Step 7.** Setup script M-file to read/write values from/to SysML instance model*

Since MATLAB scripts do not have input and output arguments, users must add commands to the beginning/end of the script to read/write values from/to SysML instance model. Follow the steps below to setup your script M-file to read values from **input.txt** file and write results to **output.txt** file.

1) Add commands to achieve the following at the beginning of your script M-file
   a) Define an array variable in the MATLAB script that will hold values of input parameters.
   b) Load the input.txt file to populate this variable.
   c) Assign the values to variables in the script

   For example, a variable **insel** is defined to contain values loaded from **input.txt** file. Then, four values contained in the **insel** variable are assigned to four variables in the script.

   ```
   inSel= load('input.txt');

   o1=inSel(1);
   o2=inSel(2);
   TempOutsite=inSel(3);
   Amplitute=inSel(4);
   ```

   Note that the order in which the values are written to the **input.txt** file is the order in which input parameters are listed in the constraint specification of the constraint block. As an example, for the constraint block in Figure 69, the **input.txt** will contain values of the variable in the following order: **row**, **col**, **outtemp**, **daycyc**.

2) Add a **save** command at the end of your script M-file to save the value of the solved variable—corresponding to the output parameter of the constraint property—in **output.txt** file. For example, to save the value of variable **a**, the following command is used.

   ```
   save('output.txt','a','-ASCII');
   exit
   ```

   The **exit** command ensures that the MATLAB session ends after script execution. This will avoid having multiple sessions of MATLAB running as the SysML model is solved multiple times.

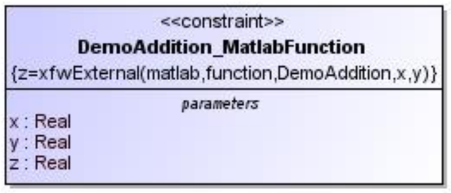
ParaMagic™ writes SysML instance values corresponding to the input parameters of a constraint property to a text file (**input.txt**) located in the same folder as the MATLAB script M-file. To import the value of the variables computed from script execution to the SysML instance model, ParaMagic reads a text file (**output.txt**) containing the variable value and located in the same folder as the MATLAB script M-file. Users do not need to worry about the **input.txt** and **output.txt** files created for transferring values between MagicDraw and MATLAB. These are automatically created and managed by ParaMagic™.

8.3.2 Using MATLAB functions

For using ParaMagic^{TM} with M-file functions, ensure that in Step 4 above, the constraint specification—for the constraint block that wraps the M-file function—uses the keyword **function** (as shown below) instead of **scriptascii**.

<out_param> = xfwExternal(matlab, function, <name_of_function_M-file>, <in_param_1>,...)

Figure 72 and Figure 73 below illustrate an example of a MATLAB function (**DemoAddition**) with 2 input parameters wrapped in a constraint block, which is then used to type constraint properties in a SysML parametric model.



<<constraint>>
DemoAddition_MatlabFunction
{z=xfwExternal(matlab,function,DemoAddition,x,y)}
parameters
x : Real
y : Real
z : Real

Figure 72: SysML constraint block to wrap MATLAB function

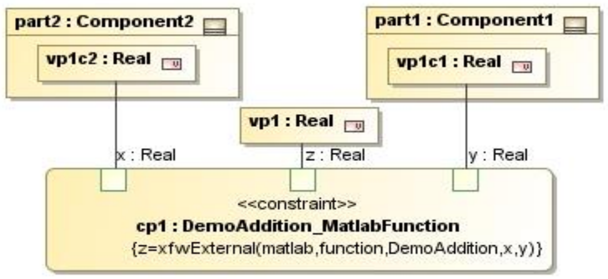


Figure 73: SysML constraint property typed by a constraint block (that wraps a MATLAB function) and used in a SysML parametric diagram

Follow the step below after completing Step 6 above.

Step 6. *Export return values to an **output.txt** file.*

Add a **save** command at the end of the MATLAB function to save the value of the output/return variable in **output.txt** file. The code snippet below shows the **save** and **exit** commands added at the end of the definition of function **DemoAddition** in a function M-file.

```
function z = DemoAddition(x,y)
z=x+y;
save('output.txt', 'z', '-ASCII')
exit
```

Note that functions can have input arguments and hence PM-MC does not require users to read values from an input.txt file (as in scripts). In the example below, the **save** command is added at the end of a function **DemoAddition** that returns the sum of two numbers. The sum is saved to **output.txt** file. As in the case of scripts, input parameters and return values of M-file functions could be single-valued or a multi-valued (e.g. array) but they cannot be complex data structures (such as an array of arrays, etc.).

9 TRADE STUDIES

With ParaMagic®, users can easily setup and run trade studies on their existing SysML models. The capability to run trade studies on SysML parametric models allows users to compute performance, reliability, cost, and other measures-of-effectiveness—especially those used to verify requirements—for a large set of system alternatives at each development phase, and then select the best-in-class alternatives for the next development phase. With ParaMagic®, trade studies can be now be performed from the earliest stages of system development. The overall process for setting up and running trade studies is as below.

1. Verify that your existing SysML instance model can be solved using ParaMagic®.
2. Setup a trade study
    a. Identify trade study inputs, outputs, and constants.
    b. Link inputs and outputs to Excel spreadsheets. ParaMagic® reads values of trade study input variables for all scenarios from linked Excel spreadsheets. After completion, the values of trade study output variables for all scenarios are written to the linked spreadsheets.
    c. Specify number of scenarios
3. Run trade study

9.1 Operation

The detailed steps for setting up and running trade studies on SysML models are as follows. The process described below assumes that you have setup a SysML schema and instance model in the same manner that you do for regular ParaMagic® solving purposes—see the Tutorials document for details.

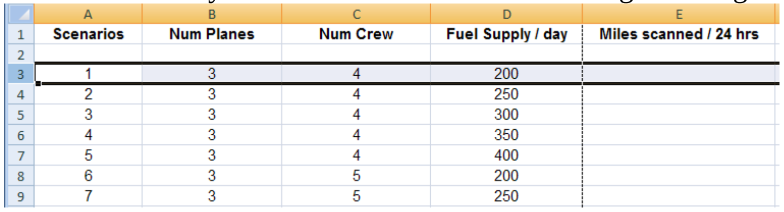
Step 1. Verify that your SysML instance model can be solved in ParaMagic®
The series of steps below are used to check if your SysML schema and instance models are structurally valid can be solved. Solving an instance model is similar to running a single scenario in a trade study.

1) *Browse the instance model*: Right click on the instance package and select **ParaMagic → Browse**. If the ParaMagic browser opens up, this implies that the schema and instance are structurally valid.
2) *Solve the instance model*: Click on the **Solve** button in the ParaMagic® browser. See section 7.1 for details. If the model solves correctly, it implies that your instance model
3) *Update SysML instance model*: Click on the **Update to SysML** button in the browser. Check that the target slot values are updated in the SysML instance model.

Step 2. Prepare Excel spreadsheet(s) with values of trade study input variables

1) Trade study variables are arranged in columns, and the scenarios are specified in rows.

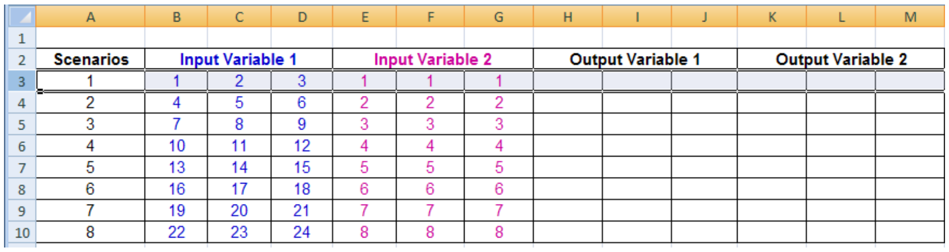
   All values of a trade study input variable should be in columns, such that each row in those columns contains values for a single scenario. In the spreadsheet shown below, **NumPlanes**, **NumCrew**, and **Fuel Supply / day** are the trade study input variables, and **Miles scanned / 24 hrs** is the output variable. Note that values for input/output variables are in columns. Each row, starting with row 3, represents the different trade study scenarios that will be solved using ParaMagic®.



	A	B	C	D	E
1	Scenarios	Num Planes	Num Crew	Fuel Supply / day	Miles scanned / 24 hrs
2					
3	1	3	4	200	
4	2	3	4	250	
5	3	3	4	300	
6	4	3	4	350	
7	5	3	4	400	
8	6	3	5	200	
9	7	3	5	250	

Figure 74: Trade study scenarios must be organized in rows—one scenario per row

For multi-valued variables (e.g. arrays), the values for each scenario should be in contiguous columns. For example, the values of **Input Variable 1** and **Input Variable 2** are arranged in columns for each scenario. Hence for scenario 1, **Input Variable 1 = {1,2,3} and Input Variable 2 = {1,1,1}.**



	A	B	C	D	E	F	G	H	I	J	K	L	M
1													
2	Scenarios	Input Variable 1			Input Variable 2			Output Variable 1			Output Variable 2		
3	1	1	2	3	1	1	1						
4	2	4	5	6	2	2	2						
5	3	7	8	9	3	3	3						
6	4	10	11	12	4	4	4						
7	5	13	14	15	5	5	5						
8	6	16	17	18	6	6	6						
9	7	19	20	21	7	7	7						
10	8	22	23	24	8	8	8						

Figure 75: Values of multi-valued variables are specified in contiguous columns for each scenario

2) Trade study variables may be linked to different workbooks/worksheets, and may have the first scenario specified in different rows for each variable, unlike as shown in the figures above.

Step 3. Connect trade study variables to Excel spreadsheets

ParaMagic uses the following logic to identify trade study input and output variables, and constants:

a) Slots with causality "**given**" and Excel access mode "**Read**" are treated as trade study input variables.
b) Slots with causality "**target**" and Excel access mode "**Write**" are treated as trade study output variables.
c) Slots with causality "**given**" but with no connection to Excel are treated as trade study constants. Hence, the value(s) specified for these slots are repeated for each trade study scenario.

Causalities were assigned to all slots in Step 1 above. In this step, you will link slots corresponding to trades study inputs and output to Excel spreadsheets. To do this, follow the steps below:

1) *Launch Excel setup*: Right click on the instance package and select **ParaMagic → Excel → Setup. This will launch the Excel setup utility, as shown below.**

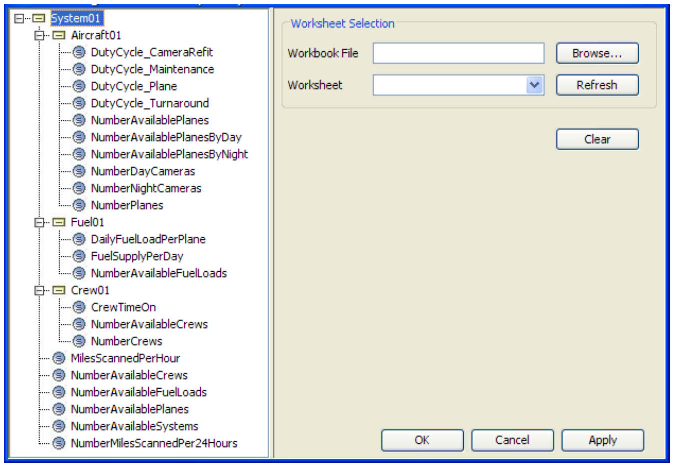
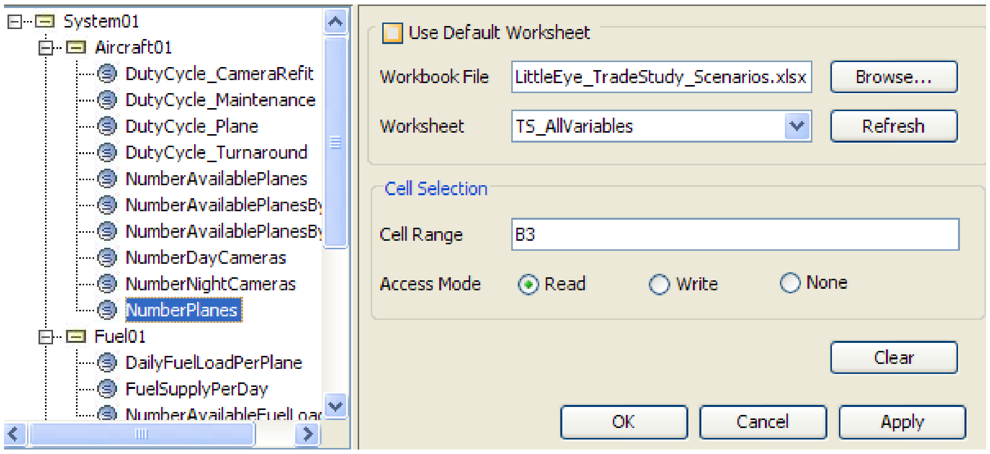


Figure 76: Use the Excel setup utility to link trade study inputs and outputs to spreadsheets

2) *Connect trade study inputs/outputs to Excel spreadsheets*: To do this, follow the steps below for each slot corresponding to a trade study input or output variable.
   a) Click on the slot in the instance tree on the left pane.
   b) Specify the Excel workbook and worksheet that contains scenario values for this slot, as shown in Figure 77 below.



Use Default Worksheet [ ]

Workbook File: LittleEye_TradeStudy_Scenarios.xlsx

Worksheet: TS_AllVariables

Cell Range: B3

Access Mode: [✓ Read] Write None

Figure 77: Use the Excel setup utility to link trade study inputs and outputs to spreadsheets

c) Set the cell range to the cell(s) that contain value(s) for the first scenario. For single-valued slots, the cell range is a single cell. For multi-valued slots, the cell range is a set of contiguous cells in a single row. As shown above, the cell range for **NumberPlanes** is set to **B3** (spreadsheet shown in Figure 74). Similarly, the cell range for **Input Variable 2** and **Output Variable 1** (spreadsheet shown in Figure 75) would be **E3:G3** and **H3:J3** respectively.

d) Set the access mode to
   i) **Read** for slots corresponding to trade study input variables.
   ii) **Write** for slots corresponding to trade study output variables.

e) Click on the **Apply** button.
   Since trade study input/output variables are setup to read/write from Excel, they are shown in blue/red color, as shown in Figure 78 below. For more details on how to use the Excel setup utility (PM-EC feature), refer to section 8.2.

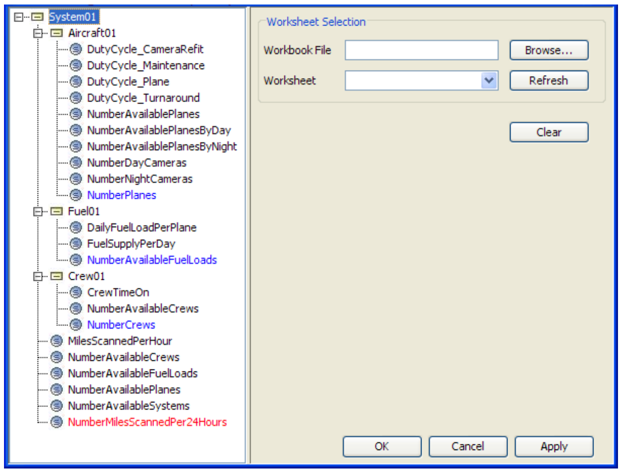


Figure 78: Slots setup to read (write) from Excel are shown in blue (red) color

Step 4. Specify number of scenarios

Right click on the instance package and select **ParaMagic** → **Trade Study** → **Setup**. Specify the number of scenarios in the dialog box, as shown below. The value (say **n**) specified for the number of scenarios will be used by ParaMagic® to construct **n** scenarios by reading the values in **n** rows (in spreadsheets connected to input variables) starting with the first row specified for each input variable.

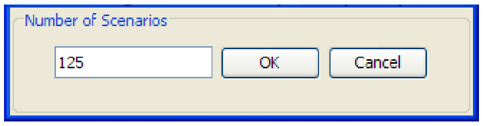
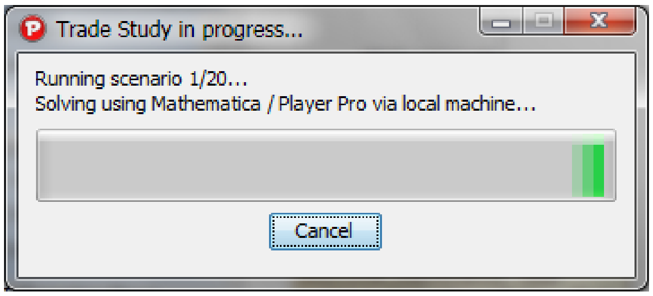


Figure 79: Specify number of scenarios for a trade study

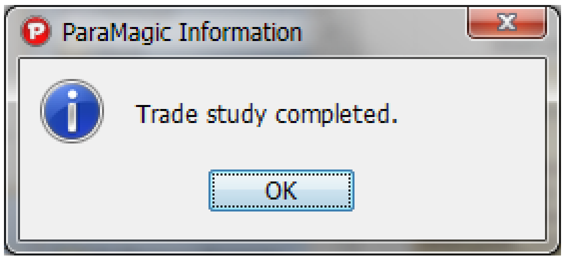
Step 5. Run trade study

Ensure that all spreadsheets connected to trade study output variables are closed. Then, right click on the instance package and select **ParaMagic** → **Trade Study** → **Run**. The trade study progress window (as shown in Figure 80) indicates the specific scenario being run and the core solver being used.



Running scenario 1/20...
Solving using Mathematica / Player Pro via local machine...

Figure 80: Trade study progress window



Trade study completed.

Figure 81: Message to indicate completion of a trade study

Completion of a trade study is indicated by the message above (Figure 81).

Step 6. View trade study outputs and perform post-processing

Open spreadsheets connected to trade study output variables to see results. You can use Excel for post-processing the values, such as for computing statistical metrics or plotting output variables against input variables.

Note that trade studies can be performed with either Mathematica or OpenModelica as a core solver. SysML parametric models executed in trade studies may include all types of relations as solved using regular ParaMagic® operation except for custom Mathematica relations that create plots for each scenario[21].

Parametric models executed during trade studies could be using constraint blocks wrapping MATLAB M-files (section 8.3) and custom-defined Mathematica functions (cMathematica – section 8.1).

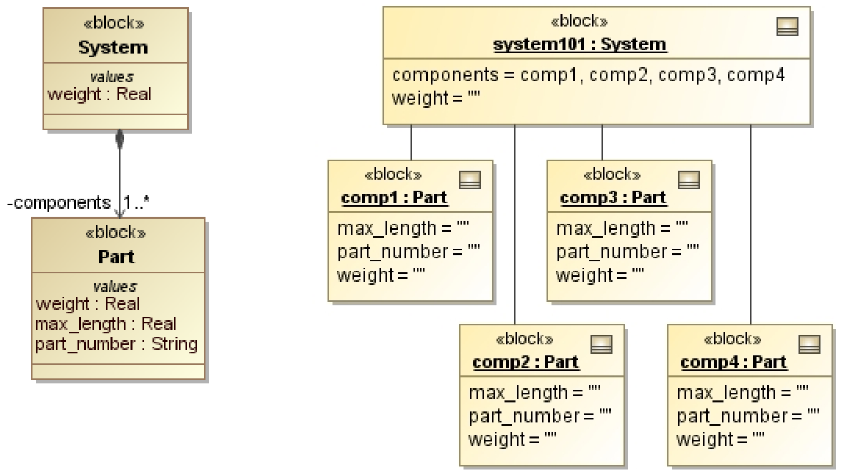
[21] Plots created for a scenario will overwrite those created for the previous scenario.

Copyright © 2014 InterCAX LLC. All Rights Reserved.

9.2 Limitations

The trade study capability in ParaMagic® has the following limitations:

1) A trade study is based on a SysML instance model which represents the structure of all scenarios. The scenarios may differ in the values assigned to the slots but not the number of instances in the SysML instance model. Figure 82 illustrates a simple SysML schema model that represents a system composed of 1 or more parts. The figure also shows a SysML instance model that conforms to the schema model. The instance model represents a system (**system101**) with 4 components. If one were to setup a trade study for this example, it would be setup for the instance model. All scenarios will represent systems with 4 components. The scenarios may vary in slot values, i.e. the values of the slots **max_length**, **part_number**, and **weight** (of the component instances **comp1**, **comp2**, **comp3**, and **comp4**) may be different in the scenarios.



SysML Schema SysML Instance structure

Figure 82: Trade studies in ParaMagic™ work on a fixed SysML instance structure

2) Values of trade study input variables must be explicitly specified for all scenarios in Excel spreadsheets. Specification of values as intervals and automated generation of scenarios by creating combinations of these intervals is not supported in this version of ParaMagic®.

3) Trade study runs are functionally similar to batch execution of a set of pre-defined scenarios. Automated generation of scenarios based on techniques to explore the design space is not supported. Contact us (info@intercax.com) for tailored interfaces to commercial tools, such as Isight[22] and ModelCenter[23], that provide design space exploration and optimization capabilities.

4) Plotting capabilities, such as the generation of single factor plots, interaction effects matrix plots, and carpet plots for trade studies, are not available natively with this version of ParaMagic®. Since trade study outputs are written to Excel spreadsheets, users may leverage the extensive plotting and post-processing capabilities of Excel.

[22] Isight: http://www.simulia.com/products/isight.html
[23] ModelCenter: http://www.phoenix-int.com/software/phx_modelcenter.php

10 PARAMAGIC® SILENT (NEW)

The ParaMagic® Silent feature is being introduced in this version (ParaMagic® 18.0). **It allows users to invoke ParaMagic® in silent mode from their MagicDraw plugins if they have ParaMagic® plugin installed with a valid license**. The ParaMagic® Silent feature provides two main automation capabilities, as listed in sections 10.1 and 10.2 below.

10.1 Invoking ParaMagic® in silent mode from plugins

This capability allows users to invoke ParaMagic® in silent mode from their own MagicDraw plugins. Here, "silent mode" implies that ParaMagic® can be used to validate and solve SysML parametric models, and update the SysML instance model with solved results without launching the ParaMagic® browser UI.

10.1.1 Java API calls to invoke ParaMagic® Silent

To invoke ParaMagic® in silent mode from your MagicDraw plugins, follow the steps below. A demo plugin with source code is provided with ParaMagic® to illustrate this feature. See section 10.1.2 for details.

(1) Add the jar files in MagicDraw lib folder (**<MD_Root>\lib**) to your classpath. You may have already done this for your plugin/script development.

(2) Add the **icax_paramagic.jar** file to your classpath. This file is located in the ParaMagic® plugin installation folder, e.g. **<MD_Root>\plugins\com.intercax.paramagic\icax_paramagic.jar**

(3) Add the **pb_logging.xml** file under your source (src) folder. This file is the log4j configuration for logging messages when running ParaMagic® Silent and is provided with the demo plugin (see section 10.1.2).

(4) In your code, import the **ParaMagic_Silent** class, as below:
import com.intercax.paramagic.plugin.slient.ParaMagic_Silent;

(5) The **ParaMagic_Silent** class provides the following three *static* methods.

com.intercax.paramagic.plugin.slient.ParaMagic_Silent

public class ParaMagic_Silent *extends* **java.lang.Object**

Constructors:
ParaMagic_Silent()

Methods:

Modifier and Type	Method and Description
static boolean	**validate(com.nomagic.uml2.ext.magicdraw.classes.mdkernel.InstanceSpecification is, java.io.File logFile, int doneCheckingIntervalInMillSeconds)** When invoked on a SysML instance specification, this method validates the instance structure, associated block structure, and parametric models.

	Parameters **is** SysML instance specification that is validated **logFile** file where log messages are written out. If null, log messages will be written out to the paramagic_silent.log file in the ParaMagic working folder, e.g. C:\Users\<Your Account>\AppData\Local\.magicdraw\18.0\plugins\com.intercax.paramagic on Windows 7 **doneCheckingIntervalInMillSeconds** time interval in milliseconds after which ParaMagic continuously checks if the validation is done. **Returns** **true** if validation is successful, **false** otherwise
static java.util.Map<java.lang.String, java.lang.Object>	**validateAndSolve(com.nomagic.uml2.ext.magicdraw.classes.mdkernel.InstanceSpecification is, java.io.File logFile, int doneCheckingIntervalInMillSeconds)** When invoked on a SysML instance specification, this method validates the instance structure and the associated block structure, and solves the parametric model for that instance. **Parameters** **is** SysML instance specification that is validated and for which the parametric model is solved. When working with ParaMagic® in plugin mode, this is the instance for which you launch the solve action (ParaMagic > Browse, press Solve). **logFile** file where log messages are written out. If null, log messages will be written out to the paramagic_silent.log file in the ParaMagic working folder, e.g. C:\Users\<Your Account>\AppData\Local\.magicdraw\18.0\plugins\com.intercax.paramagic on Windows 7 **doneCheckingIntervalInMillSeconds** time interval in milliseconds after which ParaMagic continuously checks if the solving is done and results are available. **Returns** Map whose keys are the MagicDraw ids of the instance slots, and values are solved values resulting from parametric model execution. The map includes

	ids of only those slots that have been solved (causality=target/ancillary). For single-valued primitive slots, the value (Object) is a Number object Number. A null value indicates that the solving was unsuccessful. For multi-valued (array) primitive slots, the value (Object) is a List of Number objects and includes all values irrespective of their causality (target, ancillary, undefined, or given). A null value in the list either represents a slot value that was not required to be solved (causality=undefined) or could not be solved (causality=target). If the map is null or empty, it indicates that there was an error, such as due to license validation, model validation, or solving. Refer to the logFile for details.
static java.util.Map<java.lang.String, java.lang.Object>	**validateAndSolveAndUpdate(com.nomagic.uml2.ext.magicdraw.classes.mdkernel.InstanceSpecification is, java.io.File logFile, int doneCheckingIntervalInMillSeconds)** When invoked on a SysML instance specification, this method validates the instance structure, associated block structure, and parametric models; solves the parametric model for that instance; and updates the SysML instance model with the solved results. **Parameters** **is** SysML instance specification that is validated and for which the parametric model is solved. When working with ParaMagic® in plugin mode, this is the instance for which you launch the browse, solve, and update action (ParaMagic > Browse, press Solve, press Update to SysML). **logFile** file where log messages are written out. If null, log messages will be written out to the paramagic_silent.log file in the ParaMagic working folder, e.g. C:\Users\<Your Account>\AppData\Local\.magicdraw\18.0\plugins\com.intercax.paramagic on Windows 7 **doneCheckingIntervalInMillSeconds** time interval in milliseconds after which ParaMagic continuously checks if the solving is done and results are available. **Returns** Map whose keys are the MagicDraw ids of the instance slots, and values are solved values resulting from parametric model execution. The map includes ids of only those slots that have been solved (causality=target/ancillary). For single-valued primitive slots, the value (Object) is a Number object Number. A null value indicates that the solving was unsuccessful.

	For multi-valued (array) primitive slots, the value (Object) is a List of Number objects and includes all values irrespective of their causality (target, ancillary, undefined, or given). A null value in the list either represents a slot value that was not required to be solved (causality=undefined) or could not be solved (causality=target). If the map is null or empty, it indicates that there was an error, such as due to license validation, model validation, or solving. Refer to the logFile for details.
static boolean	**validateAndRunTradeStudy(com.nomagic.uml2.ext.magicdraw.classes.mdkernel.InstanceSpecification is, java.io.File logFile, java.io.File resultFile, int doneCheckingIntervalInMillSeconds)** When invoked on a SysML instance specification that is setup for trade studies, this method validates the instance structure, associated block structure, and parametric models; and runs the trade study for that instance. **Parameters** **is** SysML instance specification that is used as a template for running the trade study. When working with ParaMagic® in plugin mode, this is the instance for which you launch the trade study action (ParaMagic > Trade Study > Run). **logFile** file where log messages are written out. If null, log messages will be written out to the paramagic_silent.log file in the ParaMagic working folder, e.g. C:\Users\<Your Account>\AppData\Local\.magicdraw\18.0\plugins\com.intercax.paramagic on Windows 7 **doneCheckingIntervalInMillSeconds** time interval in milliseconds after which ParaMagic continuously checks if the trade study is done. **Returns** **true** if successful in running the trade study, **false** otherwise

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

(6) If your ParaMagic® license has expired and you try to invoke the static methods in **ParaMagic_Silent**, they will return **false** and you will see the following message written out to the ParaMagic Silent log file.

```
2014-08-07 16:42:40 WARN  - ParaMagic plugin license has expired. Contact your administrator or No Magic to renew your license.
```

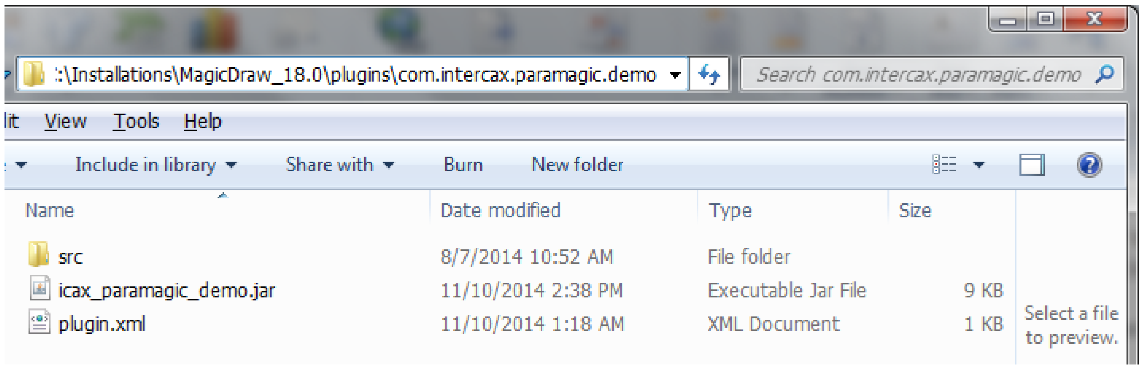
10.1.2 Install, Test, and Review the ParaMagic Silent Demo plugin

A demo plugin with source code is provided with ParaMagic® to illustrate how to invoke the ParaMagic® Silent feature. It is located in the **Other Examples** folder (section 5.3) after the ParaMagic® plugin installation:

- Folder: **<MD_Root>\samples\ParaMagic\Other_Examples\**
- File: **com.intercax.paramagic.demo.zip**

Follow the steps below to review and test the demo plugin.

1) Extract the demo plugin zip file - **com.intercax.paramagic.demo.zip**. It will create a new folder with the same name (**com.intercax.paramagic.demo**) which will have the following contents.



:\Installations\MagicDraw_18.0\plugins\com.intercax.paramagic.demo
Search com.intercax.paramagic.demo
lit View Tools Help
Include in library
Share with
Burn
New folder
Select a file to preview.

Name	Date modified	Type	Size
src	8/7/2014 10:52 AM	File folder	
icax_paramagic_demo.jar	11/10/2014 2:38 PM	Executable Jar File	9 KB
plugin.xml	11/10/2014 1:18 AM	XML Document	1 KB

Figure 83: Contents of the ParaMagic Silent Demo plugin

2) To view the source code, see the **src** folder. It includes the following:
   a) **pb_logging.xml** file is the log4j configuration file that is need for configuring ParaMagic Silent logging. It should be included in the src folder of your MagicDraw plugins and scripts.
   b) **ParaMagicDemoPlugin.java** class under **com\intercax\paramagic\demo\plugin** folder. This single class has the plugin definition, including the actions. See the nested class **ParaMagic_SilentThread** that is instantiated and run from the **actionPerformed**(..) method of **ParaMagicSilentAction** class

3) Copy the folder **com.intercax.paramagic.demo** to the MagicDraw plugins directory, e.g. **<MD_Root>\plugins.** Close and restart MagicDraw as an Administrator. This will install the demo plugin.

4) Open any sample model that comes with ParaMagic®, e.g. LittleEye model in the Tutorials folder. Right click on the **coverageAnalysis** instance and you can see ParaMagic Silent Demo plugin with two menus.

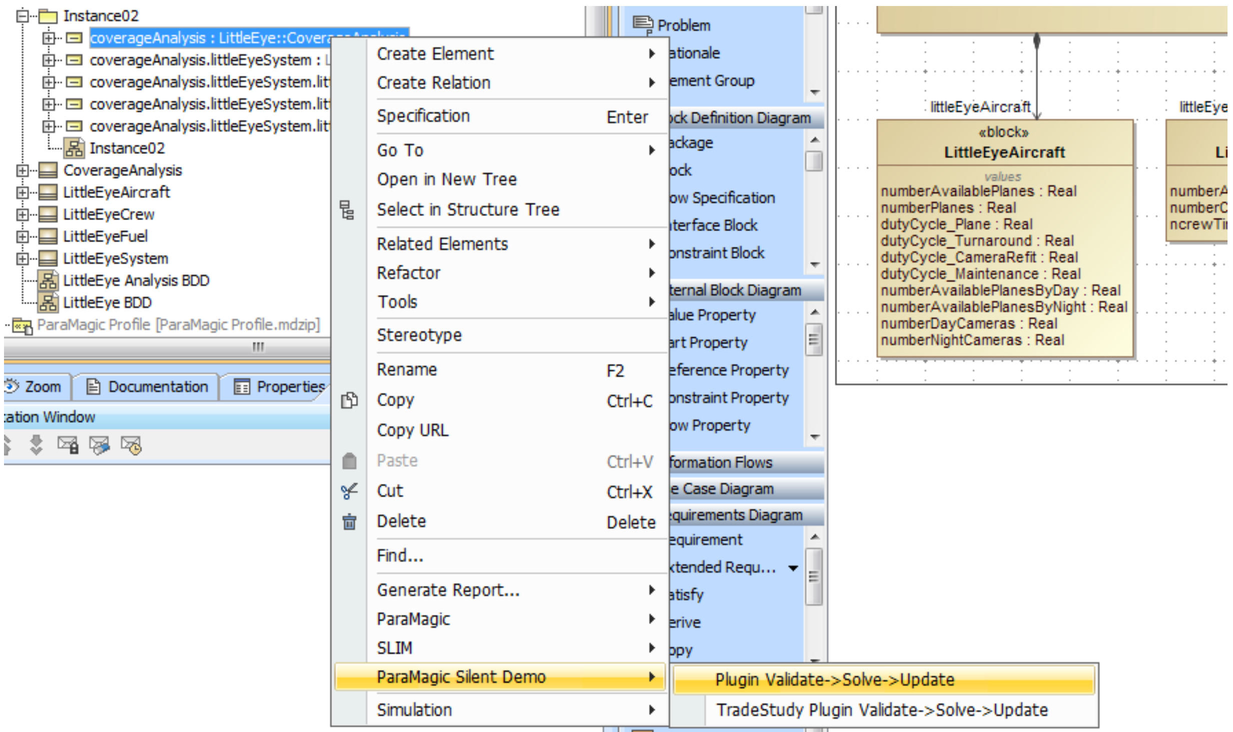
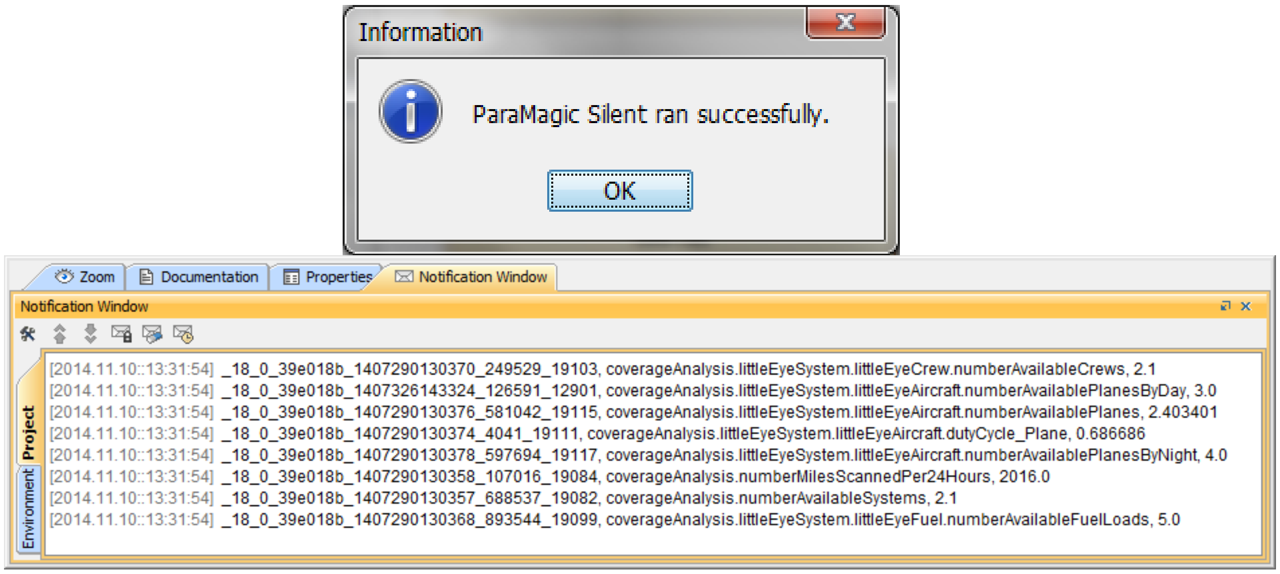


Figure 84: Demo plugin to illustrate the use ParaMagic® Silent feature

a) **Plugin Validate->Solve->Update** will invoke the **validateAndSolveAndUpdate(…)** method listed in section 10.1.1 for a given instance.

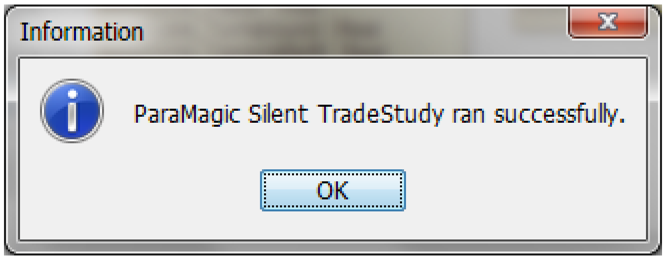
If the solving is successful, an information window will indicate so (Figure 85 – top). Once the OK button is pressed, results will be written out to the MagicDraw notification window, as shown below (Figure 85 - bottom), in the format **<slot id>, <slot name>, <slot value after solving>**. For example, try invoking the menu on the **LittleEye::Instance02::coverageAnalysis** instance in the **LittleEye** model available under the ParaMagic® tutorials.



```
[2014.11.10::13:31:54] _18_0_39e018b_1407290130370_249529_19103, coverageAnalysis.littleEyeSystem.littleEyeCrew.numberAvailableCrews, 2.1
[2014.11.10::13:31:54] _18_0_39e018b_1407326143324_126591_12901, coverageAnalysis.littleEyeSystem.littleEyeAircraft.numberAvailablePlanesByDay, 3.0
[2014.11.10::13:31:54] _18_0_39e018b_1407290130376_581042_19115, coverageAnalysis.littleEyeSystem.littleEyeAircraft.numberAvailablePlanes, 2.403401
[2014.11.10::13:31:54] _18_0_39e018b_1407290130374_4041_19111, coverageAnalysis.littleEyeSystem.littleEyeAircraft.dutyCycle_Plane, 0.686686
[2014.11.10::13:31:54] _18_0_39e018b_1407290130378_597694_19117, coverageAnalysis.littleEyeSystem.littleEyeAircraft.numberAvailablePlanesByNight, 4.0
[2014.11.10::13:31:54] _18_0_39e018b_1407290130358_107016_19084, coverageAnalysis.numberMilesScannedPer24Hours, 2016.0
[2014.11.10::13:31:54] _18_0_39e018b_1407290130357_688537_19082, coverageAnalysis.numberAvailableSystems, 2.1
[2014.11.10::13:31:54] _18_0_39e018b_1407290130368_893544_19099, coverageAnalysis.littleEyeSystem.littleEyeFuel.numberAvailableFuelLoads, 5.0
```

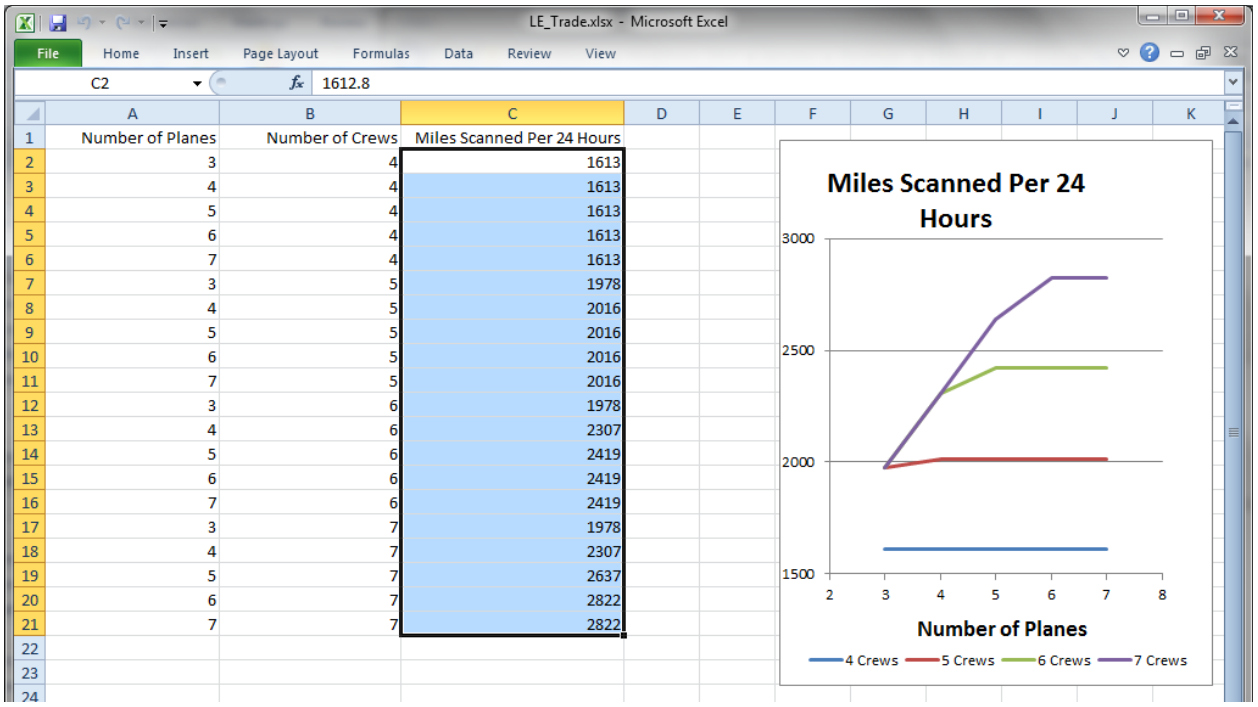
Figure 85: Result of invoking Plugin Validate->Solve->Update menu on the coverageAnalysis instance in the LittleEye tutorial model.

b) **TradeStudy Plugin Validate->Solve->Update** will invoke the **validateAndRunTradeStudy(...)** method listed in section 10.1.1 for a given instance that has been setup to run trade studies. For example, try invoking the menu for the **LittleEye::Instance02::coverageAnalysis** instance in the **LittleEyeTradeStudy** model available under ParaMagic® tutorials. The following message (Figure 86) will be shown and the trade study results will be written out to the Excel spreadsheet setup with the instance model—**LE_Trade.xlsx** available with the **LittleEyeTradeStudy** model (Figure 87). For more details on ParaMagic® trade studies, refer to section 9.



ParaMagic Silent TradeStudy ran successfully.

Figure 86: Message indicating the successful completion of trade study run.



	A	B	C
1	Number of Planes	Number of Crews	Miles Scanned Per 24 Hours
2	3	4	1613
3	4	4	1613
4	5	4	1613
5	6	4	1613
6	7	4	1613
7	3	5	1978
8	4	5	2016
9	5	5	2016
10	6	5	2016
11	7	5	2016
12	3	6	1978
13	4	6	2307
14	5	6	2419
15	6	6	2419
16	7	6	2419
17	3	7	1978
18	4	7	2307
19	5	7	2637
20	6	7	2822
21	7	7	2822

Figure 87: Trade study results are written out to the Excel spreadsheet setup with the trade study instance model.

10.2 Invoking ParaMagic® in silent mode from the ParaMagic® plugin

ParaMagic® can be invoked in silent mode from the ParaMagic® plugin itself. This is useful when users want to solve and update the SysML instance model with a single mouse click without launching the ParaMagic® Browser. See the command menus: **ParaMagic > Util > Silent > Solve** and **ParaMagic > Util > Silent > Solve and Update** described in section 7.1.

11 COPYRIGHT

11.1 Copyright statement from InterCAX LLC

This User Guide and the software described therein are copyrighted. No part of this user guide or the described software may be copied, reproduced, translated, or reduced to any electronic medium or machine-readable form without the prior written consent of InterCAX LLC.

11.2 Liability disclaimer from InterCAX LLC

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNERS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.