InterCAX LLC
75 Fifth Street NW, Suite 213
Atlanta, GA 30308, USA
Voice: +1- 404-592-6897
Web: www.InterCAX.com/melody
E-mail: melody@intercax.com

Melody™ R3

SysML Parametric Solver for IBM Rational Rhapsody

Users Guide

Table of Contents

1 ABOUT

This is the Users Guide for Melody™ Release 3 (R3) for IBM Rational Rhapsody. The sections in this document and their purpose are stated below:

- Updates since the last release (Melody™ R2) – lists the new features and improvements in Melody™ R3.
- Quick Start – speed up learning Melody™ (run existing models, create new models)
- Installation – installation instructions for Melody™
- User Documents – important documents for using Melody™ (including tutorials and examples)
- SysML Model Requirements – requirements for creating SysML models executable in Melody.
- Program Features – learn about Melody's user interface
- Connections to External Tools – learn about incorporating Mathematica functions, MATLAB functions/scripts & Simulink models, and Excel spreadsheets in your SysML parametric models and orchestrating their execution using Melody™.
- Trade Studies – learn about setting up and executing trade studies on your parametric models to select the best-in-class system alternatives.
- Copyright – important copyright information that users must read

Melody™ R3 comes in two editions – Standard and Pro. The table below compares the features of these editions. All features (both editions) are described in this Users Guide.

Features	Melody™ Editions	
	Standard	Pro
Regular math solving *Library of math functions available*	**Yes**	**Yes**
Use Mathematica or OpenModelica (free) as a core solver	**Yes**	**Yes**
Excel Connection *Connect SysML block attributes to Excel spreadsheets and read/write from/to spreadsheets.*	**Yes**	**Yes**
Trade Studies *Execute parametric models for a set of scenarios defined in Excel spreadsheets, and export trade results to spreadsheets for plotting and post-processing.*	**Yes**	**Yes**
MATLAB/Simulink Connection *Wrap MATLAB functions/scripts as constraint blocks and use in parametric models; requires Mathematica as a core solver*	No	**Yes**
Custom Mathematica Connection *Wrap custom Mathematica functions as constraint blocks and use in parametric models; requires Mathematica as a core solver*	No	**Yes**
Execute Complex Aggregate Relationships *Execute parametric models involving aggregate part / reference / shared properties*	No	**Yes**

The following conventions are used in this document:

- Melody™ implies this Melody™ R3 plugin
- Rhapsody (or Rhapsody SysML) implies IBM Rational Rhapsody Designer (or Architect) for System Engineers, version 7.5.2.
- Text enclosed by < > implies that you need to provide your system-specific settings, such as installation folder path or IP address.
- Text written in this font implies that it is a computer keyword or an abbreviation for a computer keyword.
- <Rhapsody_Root> refers to the Rhapsody installation folder. (e.g. C:\Program Files\IBM\Rational\Rhapsody\7.5.2 on a Windows machine).
- <Melody_Root> refers to the Melody installation folder <Rhapsody_Root>\Share\Profiles\Melody
- OM implies OpenModelica

Rational Rhapsody[a] is a registered trademark of IBM.
Mathematica[b] is a registered trademark of Wolfram Research, Inc.
OpenModelica[c] is a freely-available modeling and simulation tool as part of the OpenModelica Project supported by the Open Source Modelica Consortium (OSMC).

2 UPDATES SINCE THE LAST RELEASE (MELODY™ R2)

Melody™ R3 offers significant end-user improvements compared to the previous version (Melody™ R2). The key end-user improvements are as follows:

1. **Execute complex aggregate relationships** — With Melody™ R3, users can execute complex aggregate relationships. Complex aggregate relationships allow users to express parametric relations over a set of part / reference / shared properties, e.g. summation of weight and cost variables for a collection of parts. For details, refer to section 6.6.4.

2. **Reuse blocks and constraint blocks** — Melody™ R3 can work with block structures and parametric models that reuse blocks and constraint blocks. It builds on improvements in parametric modeling in Rhapsody 7.5.2 and additional workarounds to provide this feature.

3. **Support for packaging structures** — Melody™ R3 can work with blocks and instances irrespective of the packaging structure. This provides users complete flexibility in organizing libraries of blocks, constraint blocks, and block instances, and reuse them for defining and evaluating different system alternatives.

4. **Execute parametric models involving nested parts** — Melody™ R3 provides a workaround for creating parametric models involving nested parts. Users can create parametric models with arbitrarily nested part / reference / shared properties, and execute them with Melody™ R3. For details, refer to item 6.1.1 (item 6).

5. **Execute parametric models involving reference / shared properties** — Melody™ R3 provides a workaround for creating and executing parametric models involving reference and shared properties. For details, refer to item 6.1.1 (item 6).

6. **Support for inheritance** — With Melody™ R3, users can execute parametric models involving inherited value properties, value properties owned by inherited part / reference / shared properties, and inherited constraint properties.

[a] IBM Rational Rhapsody: http://www-01.ibm.com/software/awdtools/rhapsody/
[b] Mathematica (Wolfram Research): http://www.wolfram.com/
[c] OpenModelica - http://www.ida.liu.se/~pelab/modelica/OpenModelica.html

7. **Incremental testing and verification** — Melody™ R3 offers users the ability to build, test, and verify parametric models incrementally. The key enabler here is that parametric execution can be invoked for any block instance in the instance model hierarchy, e.g. part instance, sub-system instance, or system instance, by selecting the subject instance in the model tree or the instance diagram. When invoked for a block instance, Melody™ executes the parametric models for that instance and all instances populating its part, reference, and shared properties, recursively. Melody™ R3 eliminates the need to create CXS and CXI heading blocks to identify the top-level block and instance.

8. **Warning users of incompatible value / data types** — Melody™ R3 warns users if parametric models involve binding connectors that connect value properties or constraint parameters with incompatible value or data types.

9. **Utilities** — Melody™ R3 provides utilities for efficient model building and debugging. Some of the main utilities are as follows. See section 7.1 for the complete list.

    a. *Refactor binding connectors* — Rhapsody 7.5.2 locates binding connectors created in parametric diagrams (for a block) inside the parent package of the block and not in the block itself. With Melody™ R3, users can automatically move binding connectors from their current location to the context block. For details, see the command Util > Refactor Binding Connectors in section 7.1, and the Melody™ tutorials.

    b. *Instance Creation* — With Melody™ R3, users can automatically generate an instance model for a given block. This utility also creates a block definition diagram that shows the instance model. For details, see Util > Create Instance in section 7.1, and the Melody™ tutorials.

    c. *Add Instances to Aggregate Property* — With Melody™ R3, users can add instances for populating aggregate part / reference / shared properties. For details, see Util > Add Instances to Aggregate Property in section 7.1, and the Melody™ tutorials (Section 6, Step V).

3 QUICK START

This section recommends a starting point for users to quickly learn Melody™.

3.1 First Pass – execute existing models

In this first pass you will review and execute parametrics in existing SysML models per the next steps.

- The first model (Addition) is a basic model for testing Melody™ 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.

1) Install Melody™ — see installation instructions in Section 3 below.

2) Open and execute the Addition model
    a) Start Rhapsody

b) Open the Addition model located here:
<Rhapsody_Root>\Share\Profiles\Melody\models\tutorials\Addition\Addition.rpy
c) Locate the package Addition::Instance01.
d) Right click on the Gamma_Inst block instance and select Melody→Browse. This will launch the Melody™ browser, as shown below.
e) Click the Solve button. After successful solution, the value of c (target variable) should equal to 5.

Note: You should have configured a core solver during installation (Section 3). If you have not installed OpenModelica (free) or Mathematica, 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.

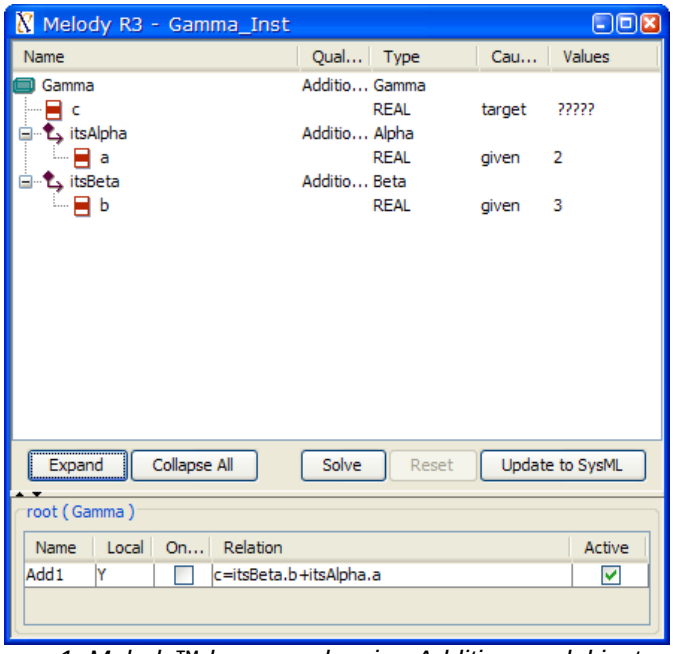


Figure 1: Melody™ browser showing Addition model instance

3) Open and execute the Satellite model
a) Open the Satellite model located here:
<Rhapsody_Root>\Share\Profiles\Melody\models\tutorials\Satellite\Satellite.rpy
b) Read input values from a spreadsheet — Right click on the package Satellite:: Satellite01 and select Melody → Excel → Read from Excel
c) Solve the model — Right click on the block instance SatelliteSystem01 located in the package Satellite:: Satellite01 and select Melody → Browse. Click on the Expand button to expand the model and then click on the Solve button. After solving, the Melody browser will show results as below (SatelliteSystem.weight = 230).

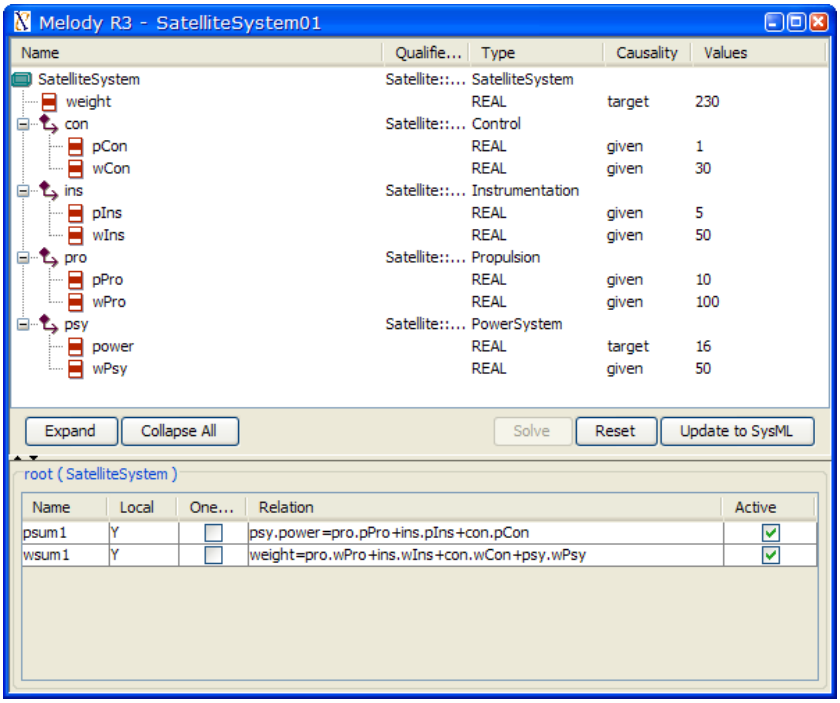


Figure 2: Melody™ browser showing Satellite model instance

d) Click on the Update to SysML button the Melody browser. This will update the SysML instance model with the solved results.

e) Write output values to a spreadsheet — Right click on the package Satellite:: Satellite01 and select Melody → Excel → Write to Excel. This will write out the Satellite Weight (230 kg) and Power Supply Required (16 KW) in the SatelliteOutput.xlsx file located in the same folder as the SysML model (<Rhapsody_Root>\Share\Profiles\Melody\models\tutorials\Satellite\).

3.2 Second Pass – create new models

This Second Pass takes you deeper, including learning how to create SysML models that contain parametrics that can be solved using Melody™. 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).
  - o This type of user needs the least amount of SysML and parametrics know-how.
  - o 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.
  - o This type of user requires more know-how.
  - o They also need a fair amount of Rhapsody SysML tool-aided modification support (in some respects more than Type 3 users).
  - o This is a good step towards becoming a Type 3 user.
- Type 3: Someone who creates their own model structures and instances from scratch (and/or from pre-existing building blocks from a library).
  - o This type of user requires a fair amount of know-how and needs good Rhapsody 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 section above provides 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 5 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: Melody has been tested to work[d] with Rhapsody installed on the following operating systems:
   a) Windows XP — 32-bit and 64-bit
   b) Windows VISTA — 32-bit
   c) Windows 7 — 64-bit
2) Java: Melody requires Java 1.6 or higher. Melody uses the same Java installation that is being used by Rhapsody. To check the Java version used by Rhapsody, follow the steps below:
   a) Open rhapsody.ini file located under <Rhapsody_Root>. For example, C:\Program Files\IBM\Rational\Rhapsody\7.5.2\rhapsody.ini
   b) Locate the JVM section in the file and ensure that the JavaLocation variable points to Java 1.6 on your computer. For example,
      [JVM]
      JavaLocation=C:\Program Files\Java\jdk1.6.0_21\
3) Hard disk space: Melody requires 50 MB of hard disk space for installation.
4) RAM: Melody requires 500 MB of memory. Additional available RAM will improve the performance of the plugin.

4.1.2 Rhapsody Requirements

1) IBM Rational Rhapsody Designer (or Architect) for System Engineers version 7.5.2 should have been installed and configured on your system. Melody R3 is not compatible with Rhapsody 7.5 or 7.5.1. It leverages updates made to SysML parametric modeling in Rhapsody 7.5.2.

4.1.3 Core Solver Requirements

Melody™ requires a core solver for backend number crunching. Users may select Mathematica or OpenModelica[e] (free) as the core solver—see section 4.2 (Step 5) for installation and configuration details.

[d] Unless otherwise specified, Melody™ may work with other editions of these operating systems but it is not been rigorously tested for editions other than those mentioned here.
[e] OpenModelica - http://www.openmodelica.org/

If you plan to use Mathematica as the core solver:

1) Melody™ has been tested with Mathematica version 7.0.
2) Melody™ has been tested to work with local Mathematica installed directly on user's hard drive or on a networked hard drive.
3) Melody™ has also been tested to work with Mathematica installed on a network, using our customized web-services product XaiTools Web Services™ (a.k.a. XWS). This allows multiple users to access a single Mathematica installation for solving services. Please contact us (melody@intercax.com) if you would like to setup XWS for your organization.

If you plan to use OpenModelica as the core solver:

1) Melody™ has been tested to work with OpenModelica 1.5.0 (on Windows) installed locally on a user's hard drive.

4.2 Installation Process

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

Step 1. Obtain a Melody™ license

After downloading Melody™, users must first obtain a license file. Evaluation licenses for Standard and Pro editions are available for a 30-day trial. After that, users must obtain a regular paid license. To request for a license, follow the steps below:

1. **Note the drive letter and volume serial number** for the volume where Rhapsody is installed on your computer. As shown in Figure 3 below, type vol at the command prompt and your drive letter and volume serial number should be displayed.

2. **Note the mac address of any of your network cards**. As shown in Figure 3 below, type ipconfig/all at the command prompt and note the physical address (mac address) of any of your network cards. For the example in Figure 3, the mac address of a wireless card is noted.

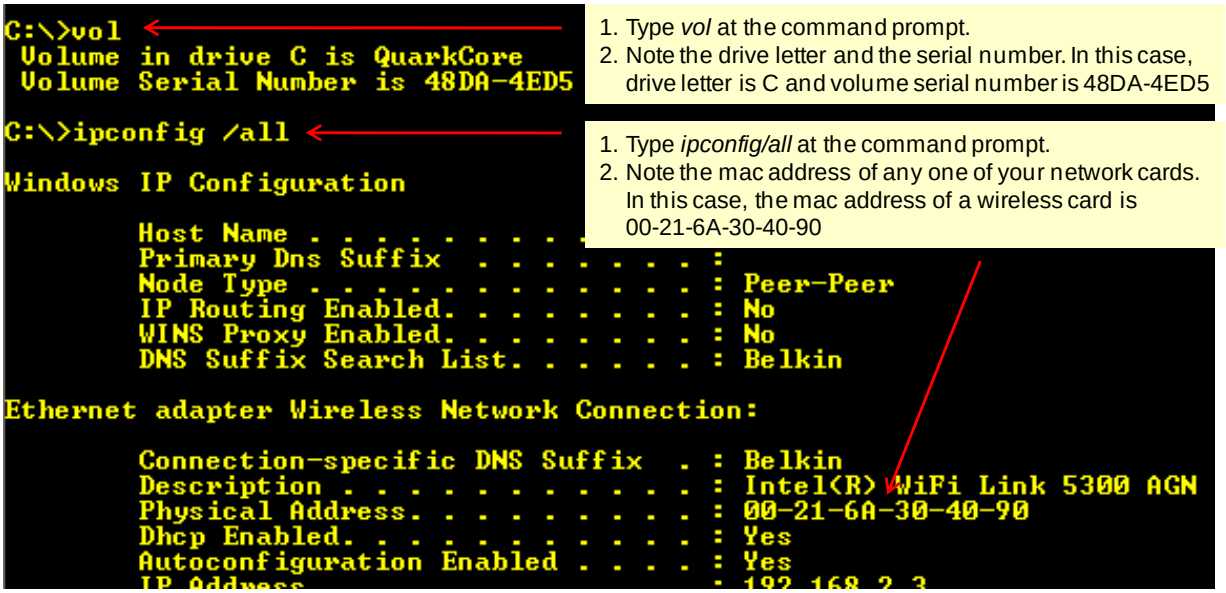


Figure 3: Information required for a Melody™ license file request

3. **Send a license request email to InterCAX**: Email the following to melody@intercax.com

a. Melody edition (Standard or Pro) for which you are requesting a license
b. Drive letter and volume serial number
c. Mac address of any of your network cards
d. Your complete contact information

4. If this is the first time you are using Melody™, we will respond within 1 business day with an evaluation license for your machine. If your evaluation period is over and you would like to purchase a regular license, contact us by phone (404-592-6897, ext 101) or email (melody@intercax.com) to setup a payment method.

5. After you obtain a Melody™ license file (Melody.lic), proceed with the installation steps below.

Step 2. Save existing projects and close Rhapsody

Step 3. Install the Melody plugin

a) Double click on the Melody™ installation executable (.exe) file.
b) Read through the license agreement and click on the I Agree button if you accept the terms and conditions.

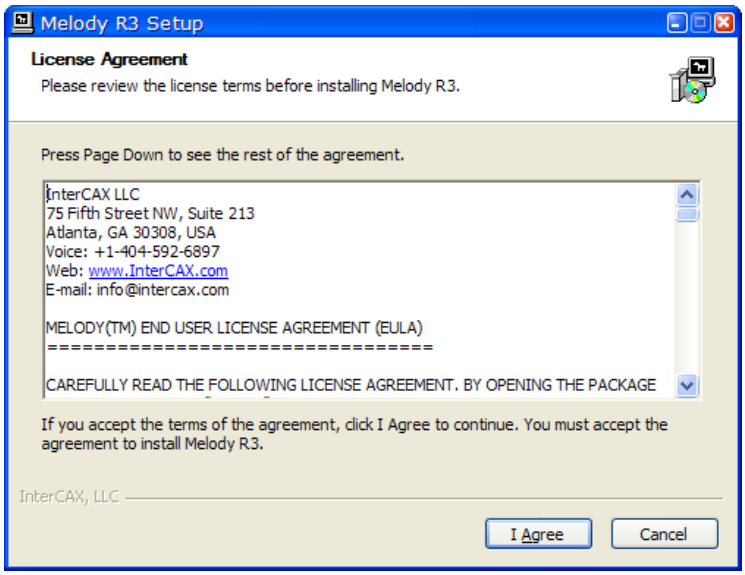


Figure 4: Melody™ installation – license agreement

c) Select the components for installation (as shown below) and click the Next button.

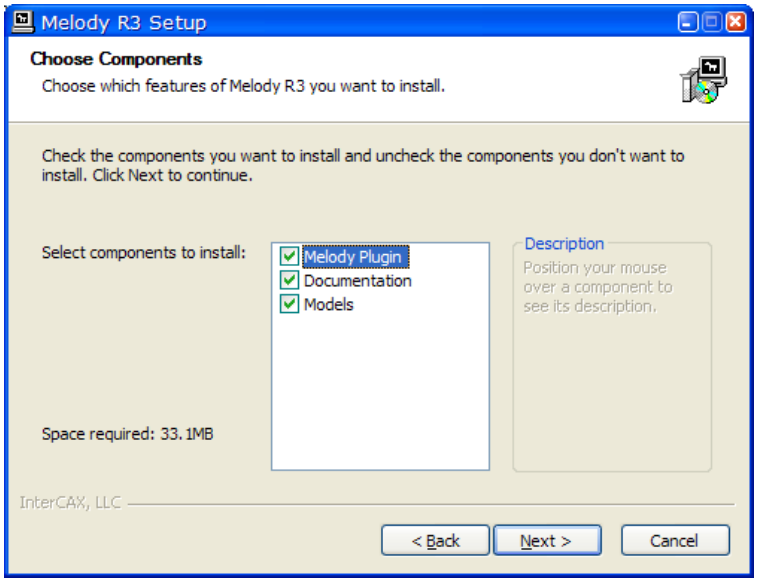
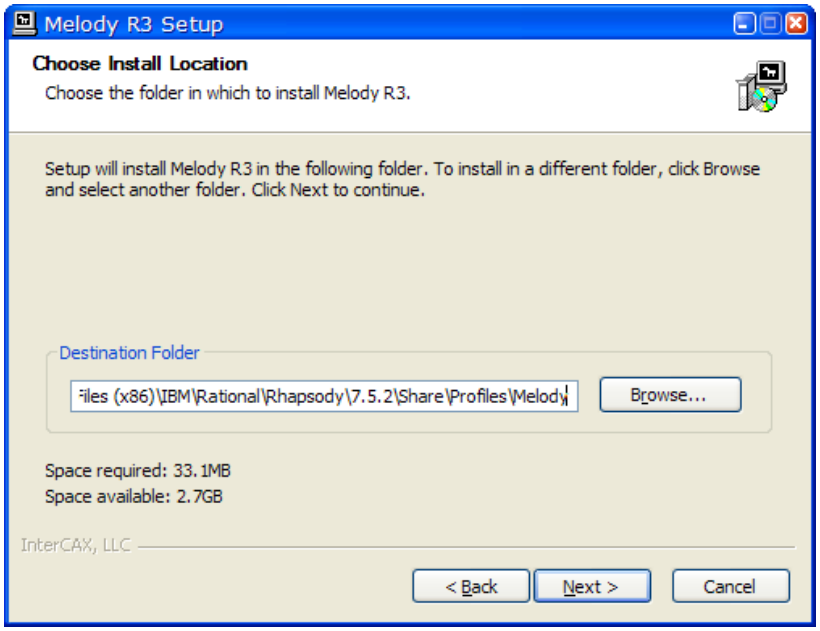


Figure 5: Melody™ installation – select all components to install

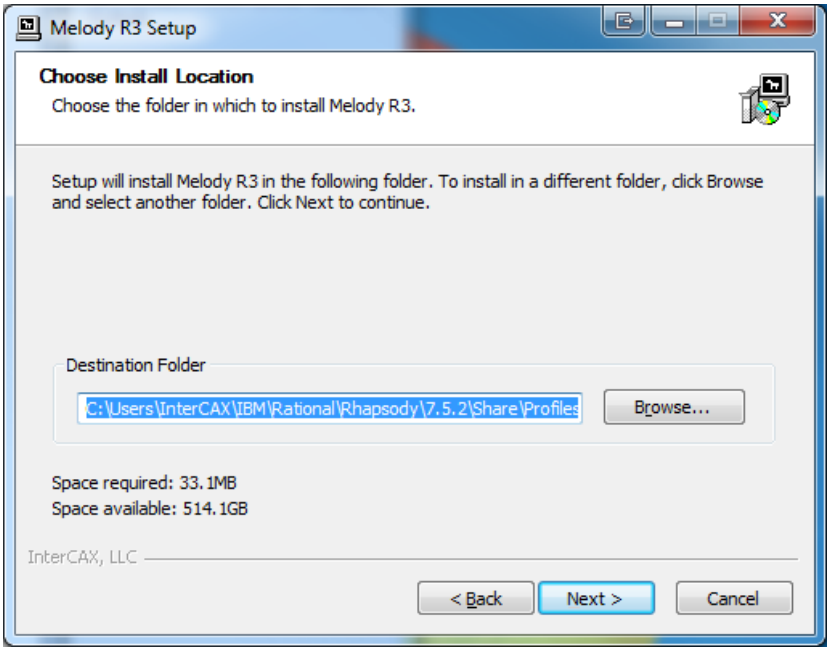
d) Select the Rhapsody installation for which you would like to install Melody™ per below:

i) For Windows XP and VISTA, select <Rhapsody_Installation>\Share\Profiles\Melody, for e.g. C:\Program Files\IBM\Rational\Rhapsody\7.5.2\Share\Profiles\Melody

ii) For Windows 7, select
C:\Users\<your account>\IBM\Rational\Rhapsody\7.5.2\Share\Profiles\Melody



Rhapsody installation / destination folder for Win XP and VISTA



Rhapsody installation / destination folder for Win 7

Figure 6: Melody™ installation folder

Note that a Melody™ installation is specific to a Rhapsody installation. If you have two versions of Rhapsody on a machine and plan to use Melody™ for both, you must install Melody™ for each of them.

e) Specify a new Start Menu folder or select an existing folder for creating Melody™ shortcuts. Alternatively, you may choose not to create shortcuts as shown below.

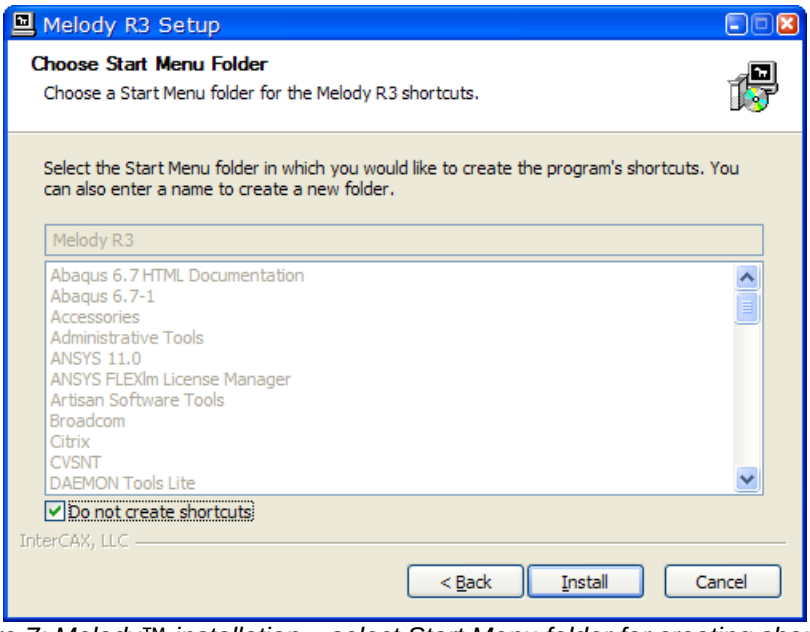


Figure 7: Melody™ installation – select Start Menu folder for creating shortcuts

f) Click on the Install button, as shown in Figure 7 above. This will start the installation process. You should see the Melody installation progress bar, as shown in Figure 8 below.

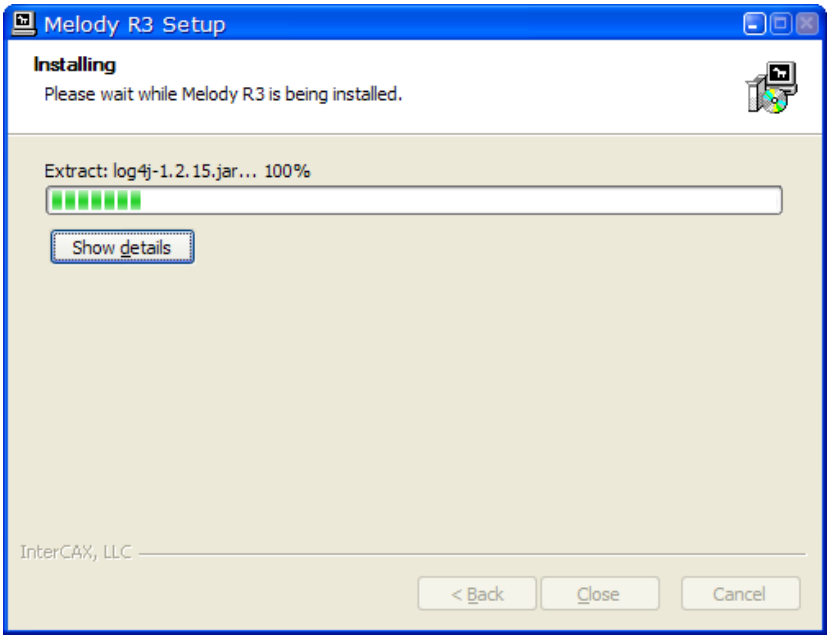


Figure 8: Melody™ installation process

Step 4. Verify installation

a) Open the Melody installation folder located here
   i) Win XP / VISTA: <Rhapsody_Root>\Share\Profiles\Melody. For example, C:\Program Files\IBM\Rational\Rhapsody\7.5.1\Share\Profiles\Melody.
   ii) Win 7: C:\Users\<your account>\IBM\Rational\Rhapsody\7.5.2\Share\Profiles\Melody

b) As shown in Figure 9 below, six folders (core, doc, lib, log, models, and xfw) and several .sbs files should have been installed.

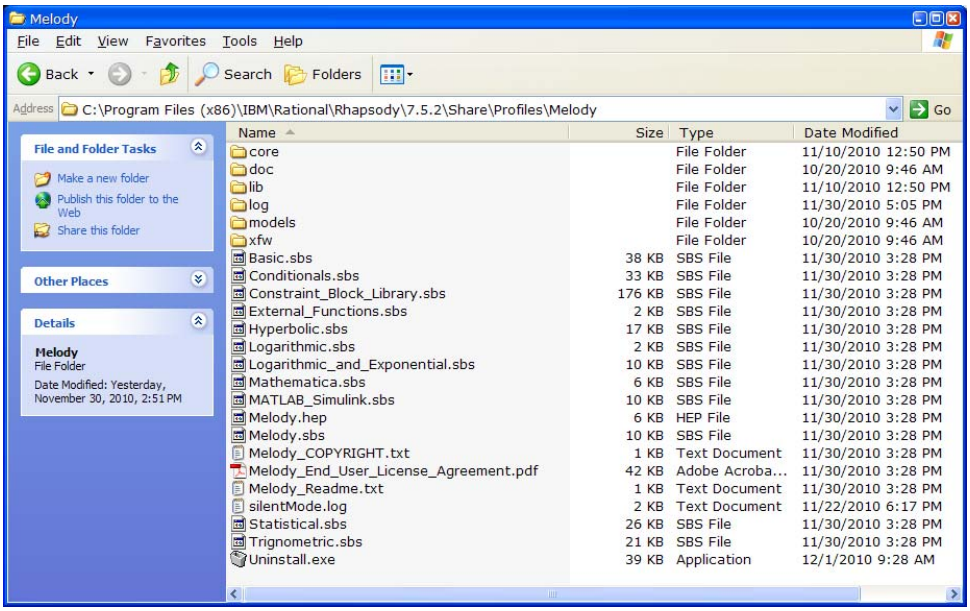


Figure 9: Melody™ installation folder after the installation process

Step 5. Select the core solver for Melody™

Melody™ allows users to select Mathematica or OpenModelica as the 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).

Specify the core solver in Melody.ini file (screenshot shown below).

a) Locate Melody.ini file in the folder <Melody_Root>\xfw\conf, for e.g. C:\Program Files\IBM\Rational\Rhapsody\7.5.2\Share\Profiles\Melody\xfw\conf\Melody.ini
b) You will see text as shown in Figure 10. Lines not starting with # character are configuration settings. The configuration variables are described in details in section 7.3.
c) Set the value of the variable com.intercax.xaitools.solver.name equal to
   1. Mathematica to use Mathematica as the core solver
   2. OpenModelica to use OpenModelica as the core solver
d) Save and close the Melody.ini file.
e) Skip Step 7 if you have selected Mathematica as the core solver, or skip Step 6 if you have selected OpenModelica as the core solver.

```
######## Melody(TM) configuration parameters ########
# (see Users Guide for further information)

#### specify name of the core solver; allowed values: Mathematica, OpenModelica
com.intercax.xaitools.solver.name=Mathematica

##### specify aspects of local tools and whether to use local or remote solver tools #####
com.intercax.xaitools.local.mathematica.true.or.false=false
com.intercax.xaitools.solver.timeout.in.seconds=180

##### specify aspects for SOAP server-based tools #####
###     Specify IP number for SOAP-based access to your Mathematica license
###     Contact us (support@magicdraw.com) to setup SOAP-based access for your Mathematica
com.intercax.xaitools.soap.mathematica.serverhost=cedar.marc.gatech.edu:8080
com.intercax.xaitools.soap.mathematica.accesskey=tempAccessMelodyKey

##### XaiTools temp location - relative to install dir ############
com.intercax.xaitools.temp.dir=temp

##### Retry interval in milliseconds for SOAP-based solver #####
RetryInvervalInMilliSeconds=1000
##### Number of retry used for soap-based solver #####
NumberOfRetry=1

##### XaiTools web service urn #####
xws.urn.version = urn:XWS_v2.2
```

Figure 10: Melody.ini file

Step 6. Mathematica installation

If you have selected Mathematica as the core solver in Step 5, you have the following choices:

- During the evaluation period (30 days), you may use our test Mathematica server—see Option 1 below.

- For long-term or production usage,
  - use Mathematica locally-installed on your machine—see Option 2a below.
  - use Mathematica installed in your enterprise as a web-service via our XWS product—see Option 2b below.

Option 1. Using our Mathematica test server for short-term evaluation

This Melody™ installation comes with a limited-time access to our Mathematica test server. Skip to step 7 and finish the installation process. Then, open and execute example models that come with this installation, per instructions in section 3.1. If Melody™ complains about an expired Mathematica license key, follows the steps below. Note that your enterprise firewall may block access to our test Mathematica server. If this is the case, use OpenModelica as a core solver (Step 7).

a) Contact us (melody@intercax.com) for Mathematica test server location and your temporary access key. After you receive your access key, follow the steps below.
b) Open the Melody.ini file.
c) Set the value of the variable com.intercax.xaitools.local.mathematica.true.or.false to false. This is set to false by default (as shown above).
d) Set the value of the variable com.intercax.xaitools.soap.mathematica.serverhost to the Mathematica test server that we provide.
e) Set the value of the access key variable com.intercax.xaitools.soap.mathematica.accesskey to the access key we provide.
f) Save and close the Melody.ini file.

After this evaluation period, you must have your own Mathematica license (Option 2a) or have access to an XWS-based server (Option 2b). Please complete the steps below to configure local / network access to your Mathematica license.

Option 2. Using your local installation of Mathematica

If you have a local installation of Mathematica, follow the steps below. Here, local implies (a) either on your machine hard drive, or (b) a networked hard drive.

a) Open the Melody.ini file.
b) Check that the com.intercax.xaitools.local.mathematica.true.or.false variable is set to true in the Melody.ini file. By default, this variable is set to false (as shown above).
c) If you are using Windows OS, ensure that Mathematica is accessible from your machine. Type math at the command line on a console. You should see some output such as below (for a Windows machine):

```
C:\>math
Mathematica 7.0 for Microsoft Windows (32-bit)
Copyright 1988-2009 Wolfram Research, Inc.
In[1]:=
```

If you do not see text similar to above, check that the environment variable “Path” has an entry pointing to the root folder where Mathematica is installed (such as C:\Program Files\Wolfram Research\Mathematica\7.0 in Windows for Mathematica 6.0 installation). If there is no such entry, add the Mathematica root folder location to the “Path” environment variable by following the steps below.

i) Right-click on My Computer, and then select Properties.
ii) Select the Advanced tab.
iii) Select Environment variables.
iv) Lookup the Path environment variable in the list of System variables
v) Select the Path environment variable and click the Edit button.

vi) Add the Mathematica root folder location at the end of the variable value field preceded by a semi-colon. If the existing entry in the variable value field is abc, then after adding the Mathematica root folder location, this field should appear as:

abc;C:\Program Files\Wolfram Research\Mathematica\7.0

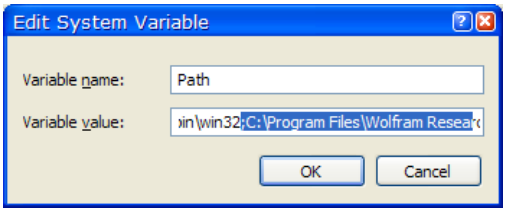


Figure 11: Adding Mathematica installation location to the Path environment variable

See the highlighted section of the Variable value field in the snapshot above.

Note: Do not replace the existing value of Variable value field in step above. Only add the Mathematica root folder location at the end of the existing value.

d) Save and close the Melody.ini file.

Option 3. Using a Mathematica installation in your enterprise as a web-service

If you have a Mathematica installation in your organization but not accessible locally, and you would like to access it remotely to use Melody, please contact us (melody@intercax.com) to help you get setup with InterCAX's web-services product (XWS). With XWS, multiple users in your organization can access Mathematica solution services and you can manage the access keys for your organization. Once you have configured XWS, follow the steps below.

a) Open the Melody.ini file.
b) Set the variable com.intercax.xaitools.local.mathematica.true.or.false to false.
c) Set the variable com.intercax.xaitools.soap.mathematica.serverhost to the IP number of the machine where XWS is installed, followed by a colon followed by the port number. For example, the variable value will look something like: 100.100.100.100:8080
d) The XWS administrator in your organization will also create an access key for you. Use this access key to set the value of the access key variable com.intercax.xaitools.soap.mathematica.accesskey.
e) Save and close the Melody.ini file.

***Note*:** You may change the settings in the Melody.ini file while Rhapsody is running. You do not need to restart Rhapsody. However, if the settings are changed while the Melody™ browser is running, you will need to close and re-launch the browser.

Step 7. OpenModelica (OM) installation

If you have selected OM as the core solver in Step 5, follow the steps below for configuring OpenModelica for Windows:

a) Download OM 1.5.0 installer (OpenModelica-1.5.0.msi file) from here: http://www.openmodelica.org/index.php/download/download-windows
b) Double click the OpenModelica-1.5.0.msi file. This will install OM on your machine. If you have an existing OM installation (older version), you will need to uninstall it first. We recommend using the latest version of OM (1.5.0).
c) Ensure that OpenModelica is specified as the core solver in Melody.ini file (Step 5 above).

d) After successfully completing Step 8 and Step 9 below, open and solve the Addition example (see section 3.1). During solving, Melody™ will indicate that it is using OpenModelica. Successful solution will indicate that OM is installed and configured to work with Melody™. Note that OpenModelica may take some extra initialization time when using it for the first time after installation.

Step 8. Place Melody™ license file in the installation folder
Place the Melody™ license file (Melody.lic) in your Melody installation folder. For example, C:\Program Files\IBM\Rational\Rhapsody\7.5.2\Share\Profiles\Melody\Melody.lic

Step 9. Start Rhapsody

Step 10. Verify installation
Follow the quick start instructions in section 3.1 (first pass). Melody™ is installed correctly if you are successful in repeating the steps in these instructions. Otherwise, contact us at melody@intercax.com.

5 USER DOCUMENTS

This installation comes with a user guide, tutorials, and other examples as described below.

5.1 Users Guide

This document is the Users Guide for Melody™ R3 plugin (both Standard and Pro editions). This Users Guide is also located here after installation:

1. Win XP and VISTA - <Rhapsody_Root>\Share\Profiles\Melody\doc\Users_Guide.pdf
2. Win 7 - C:\Users\<your account>\IBM\Rational\Rhapsody\7.5.2\Share\Profiles\Melody

5.2 Tutorials

This installation comes with 5 tutorials. Each tutorial has step-by-step instructions to create a valid SysML model that can be solved using this plugin, as described in the Tutorials.pdf document. Each tutorial also has a pre-built SysML model with instances that are ready to explore and solve. Refer to the Tutorial.pdf document on instructions to execute the models. The tutorial models and the Tutorial.pdf document are located under the models\tutorials sub-folder folder in the Melody installation folder, for e.g.
C:\Program Files\IBM\Rational\Rhapsody\7.5.2\Share\Profiles\Melody\models\tutorials\

The 5 tutorials are:

1) Addition is a SysML model for learning how to create basic parametric equations.
2) Satellite is a SysML model of a satellite system.
3) Orbital is an early-stage orbital mechanics and spacecraft model in SysML that demonstrates the use of Melody-Excel connection (section 8.1).
4) LittleEye is a SysML model of a UAV-based road scanning system named LittleEye that demonstrates the use of Melody-MATLAB connection (section 8.3) and trade study capabilities (section 9).
5) PCB is a SysML model of a printed circuit board (electronics) hardware that demonstrates the use of complex aggregate relationships and inheritance.

5.3 Other Examples

In addition to the tutorial models, this installation comes with 7 example SysML models that are ready to explore and solve. Refer to the Other_Examples.pdf document for descriptions of these models, and instructions to execute these models. These models and the Other_Examples.pdf

document are located under models\other_examples sub-folder in the Melody installation folder, for e.g. C:\Program Files\IBM\Rational\Rhapsody\7.5.2\Share\Profiles\Melody\models\other_examples\

The following example models are available with Melody™:

1) BasicSystem_cMathematica
2) BasicSystem_MATLAB
3) Bicycle
4) Energy
5) Insurance
6) Parametrics_PartRefShared_Properties
7) ProjectPlan

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.

Refer to the following publications to learn more.

1) 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/)
2) 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 Melody™ plugin. Some of these requirements are based on limitations in Rhapsody for supporting SysML parametrics and instances; some are based on recommended practices for enhanced model interoperability; some are to make SysML models less ambiguous for the plugin and solvers; while others are limitations of this version of the plugin and 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, constraint blocks, and parametrics; and an instance model conforms to this structure and has values assigned (completely or partially) to its slots (value properties). There may be several instance models that conform to a given schema. The schema represents a template for defining a family of system alternatives, while each instance model that conforms to the schema represents a specific system alternative. Representation of schema and instances enables users to define SysML parametrics model once (at the schema level) and use/execute it to compute cost, performance, and other system MoEs, verify requirements, and perform trades on for different system alternatives (instances).

6.1.1 Model schema requirements

The following requirements must be satisfied by the SysML schema model for it to be processed by Melody™.

1) Melody™ requires that all SysML elements required for defining the schema must be in packages.

2) A constraint block should have only one constraint specification. This version of the plugin does not support multiple constraint specifications defined in one constraint block.

3) All value properties (attributes) of a block must be typed by a value type (per SysML standard). Users may create their own value types or use existing value types (e.g. Real) available with the SysML profile in Rhapsody. Attributes with string values must be typed by the String value type available with the Melody™ Profile. All value properties, except for those typed by the String value type, will be expected by Melody™ to be populated with real numbers.

4) All association relationships from one block to another should be navigable only in 1 direction. Otherwise, it leads to circular references at the block level which is not handled by Melody™ - see item 5) in section 0 (Limitations).

5) Per the SysML standard,
   a) Parametric diagrams must be created inside (in the context of) a block.
   b) Binding connectors must be created inside (in the context of) a block. When a user creates a binding connector on a parametric diagram in Rhapsody 7.5.2, it is placed inside the package owning the context block and not inside the block. With Melody, users can easily relocate binding connectors for a single block or for all the blocks in a package without having to do it manually. See 7.1 for the command menu Melody→Util→Refactor Binding Connectors.

6) Creating parametric models for nested parts and reference / shared properties.

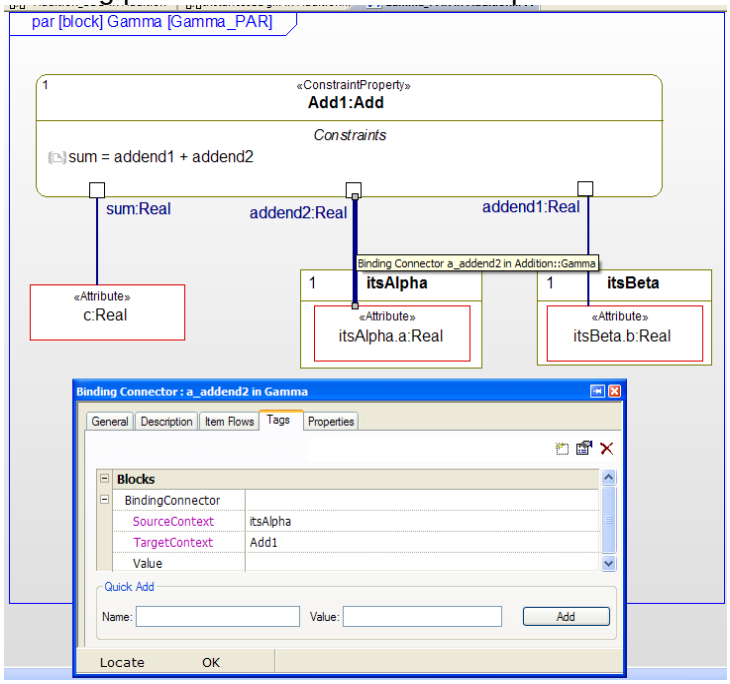


Figure 12: Rhapsody populates the end contexts of a binding connector for part properties

When a user creates a binding connector between value properties / constraint parameters on a parametric diagram, Rhapsody 7.5.2 stores the context of both the ends of the binding connector. For a given end, the context is the part property (or the constraint property) that owns the value property (or the constraint parameter) at that end of the binding connector. Figure 12 shows the SourceContext and TargetContext for the highlighted binding connector. In this case, the SourceContext is the part property (itsAlpha) that owns end1 (value property a) of the binding connector. The TargetContext is the constraint property (Add1) that owns end2 (constraint

parameter addend2) of the binding connector. If no context is specified for an end, it implies that the value property at that end of the binding connector is owned by the block in which the parametric diagram is created. Note that parametric diagrams must always be created in the context of (inside) a block per the SysML standard.

However, Rhapsody populates the end context only if the ends of a binding connector:

- do not involve nested parts
- do not involve reference or shared properties

If the ends of a binding connector involve nested parts, Rhapsody 7.5.2 does not populate the end contexts. As shown Figure 13, the source context is not populated for the highlighted binding connector. As a result, the parametric model is incomplete and ambiguous. Melody™ provides a workaround to this problem. Users can apply the BindingConnectorContext stereotype to the binding connector and use the tags (sourceContext or targetContext) to specify the missing context. As shown in Figure 13, the sourceContext tag (provided by Melody™) is set to itsA.itsA1 to indicate the nested part property owning the value property v1. The order of parts specified in the context is from outer to inner, e.g. itsA.itsA1. The SysML dot notation is used for this purpose. After a user specified the missing end context, the parametric model is complete and Melody™ can execute it.

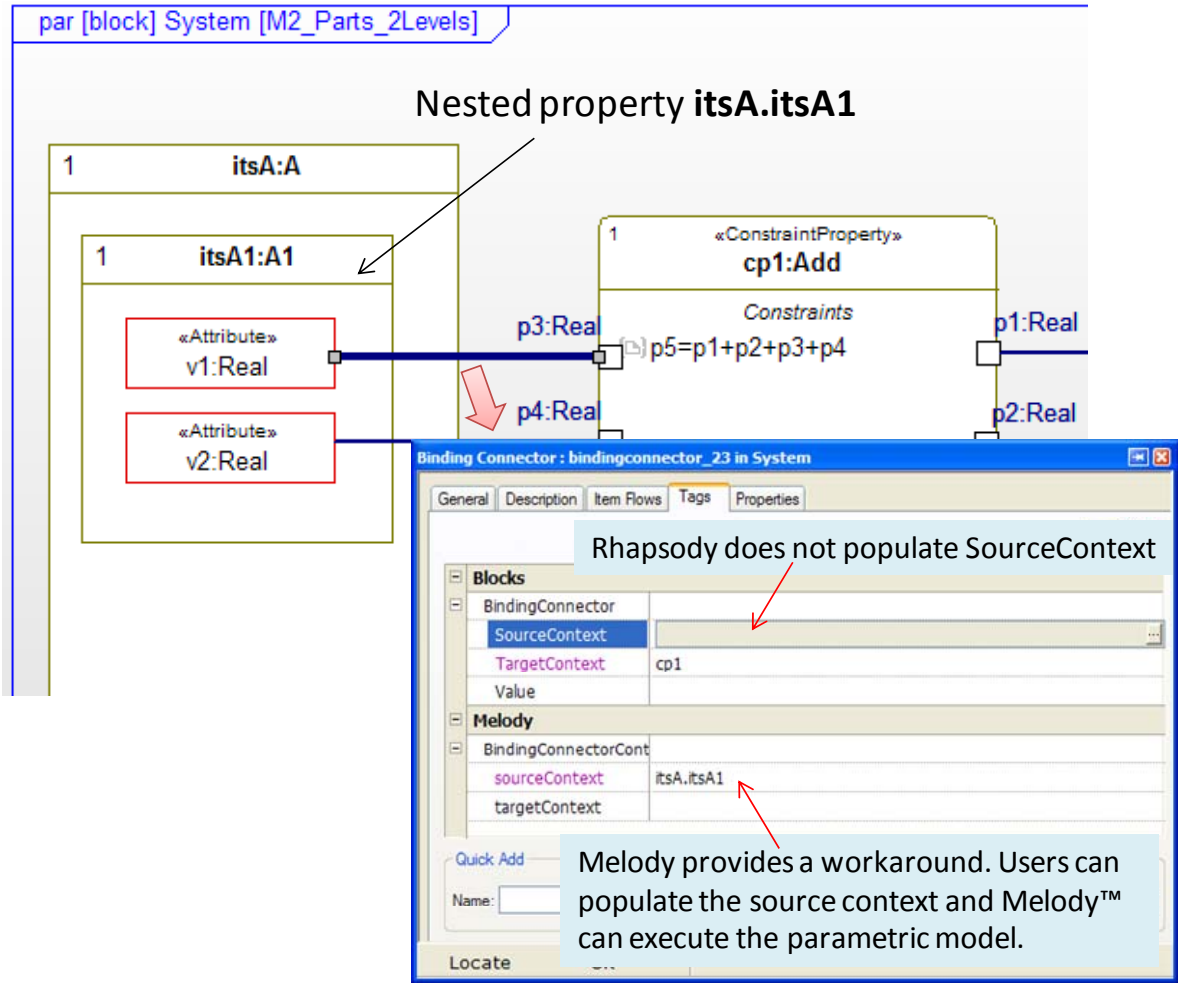


Figure 13: Melody™ provides a workaround for parametric models involving nested parts.

The same problem occurs for parametric models involving reference properties. As shown in Figure 14 below, Rhapsody does not populate the target context of the highlighted binding connector. The target context is the reference property itsB. Users can specify the target context for Melody™, using the same approach as described above. Once this is specified, the parametric model is complete and Melody™ can execute it. If the end involves nested part / reference / shared properties (or combinations), the SysML dot notation should be used (e.g. itsB.itsB1.itsB2)

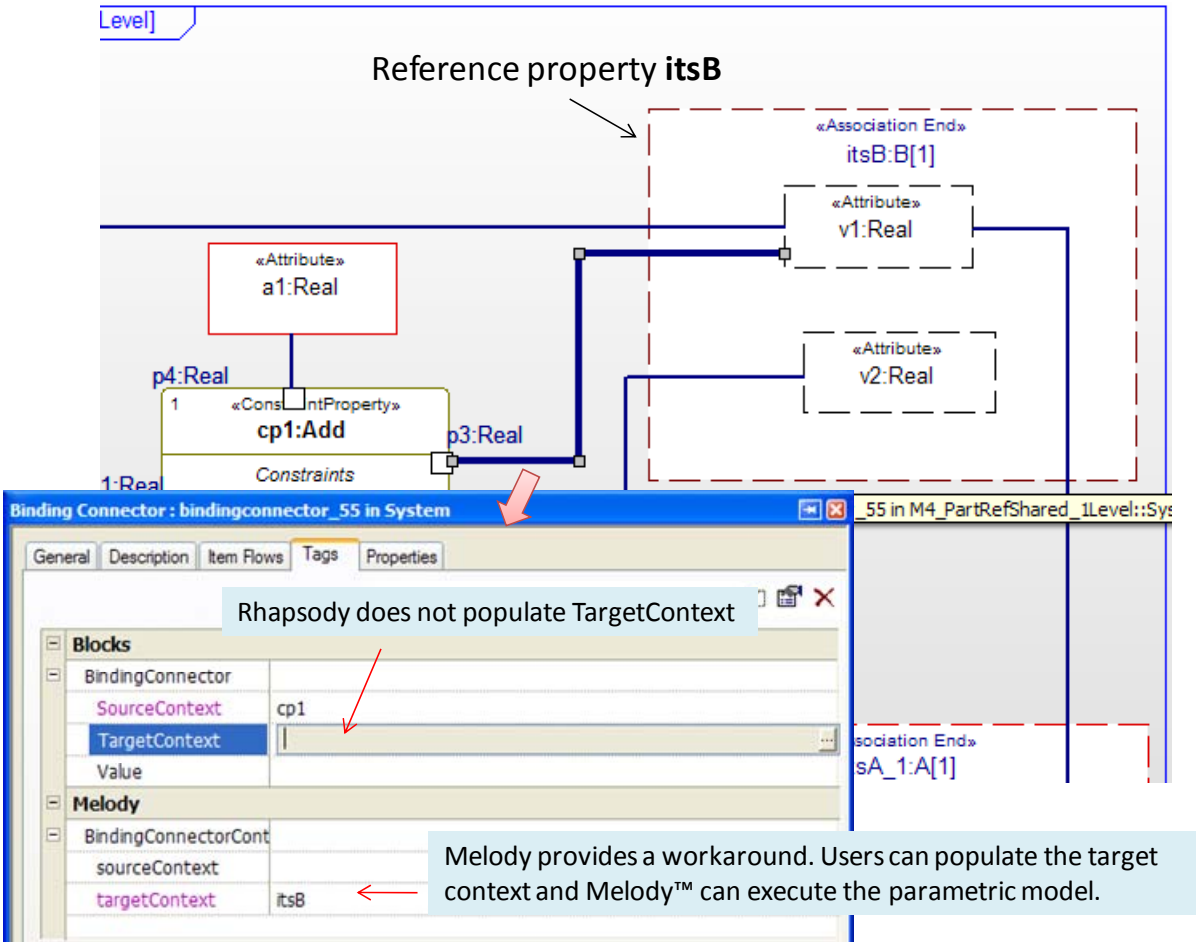
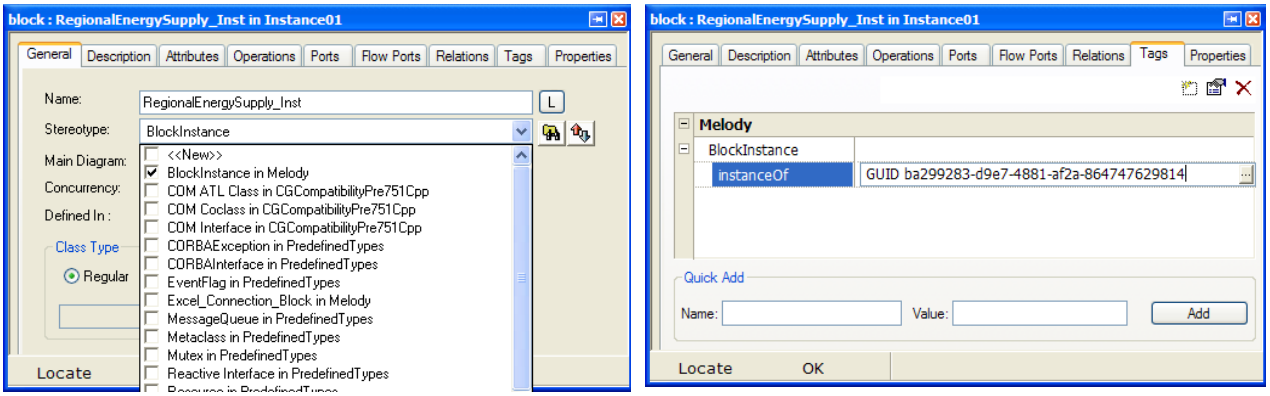


Figure 14: Melody™ provides a workaround for parametric models involving reference or shared properties

6.1.2 Model instance requirements

In SysML, block instances are represented using InstanceSpecification. Rhapsody 7.5.2 does not support SysML instances. As a workaround, Melody™ provides a special stereotype BlockInstance (available with the Melody™ profile) that can be applied to a block to identify it as a block instance, as shown in Figure 15a. A tag instanceOf available with this stereotype is used for associating an instance to the block that it conforms to. The tag is populated with the GUID of the block, as shown in Figure 15b. The Initial Value field of the value properties (attributes) is used for populating value properties.



a. Applying BlockInstance stereotype to a block to identify it as an instance.

b. The instanceOf tag is populated with the GUID of the classifier block.

Figure 15: Melody™ workaround for supporting block instances in Rhapsody

Melody™ provides capabilities to automatically generate an instance structure and diagram from a given block structure. The Util > Create Instance command (section 7.1) achieves this. This utility is described in details in each of the tutorials (Tutorials.pdf). During automated instance generation, Melody™ applies the BlockInstance stereotype and populates the instanceOf tag. If a user wants to manually create an instance or modify the classifier block, they can do so by manually populating/changing the value of the instanceOf tag (Figure 15b). The Util > Copy Element GUID command (section 7.1) can be used to obtain the GUID of a block.

The following requirements must be satisfied by the SysML instance model for it to be processed by Melody™.

1) A block instance has the same value / part / reference / shared properties as the corresponding block. Value properties are populated by specifying a value in the Initial Value field. Part / reference / shared properties are populated by setting the property type to other block instances. Block instances do not have constraint properties but are instead governed by the constraints defined in the corresponding block.

2) All the part / reference / shared properties of a block instance must be populated. Value properties with causality "given" must be populated. See the tutorials for details.

3) All value properties must be populated with plain text. For solving purposes, Melody™ checks if these values contain numbers.

4) If a value property of a block is connected to the outputs of multiple ONEWAY relations, then that corresponding value property in the block instance 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 value property slots (attributes) in a block instance. The validity of a causality assignment is based on whether or not an attribute has value(s). In the table, TRUE implies valid assignments and FALSE implies invalid assignments. For example, if an attribute has values, then its causality may be "given", "ancillary", or "target" but not "undefined". Similarly, if an attribute 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 an attribute	**given**	**undefined**	**ancillary**	**target**
has values	TRUE	FALSE	TRUE	TRUE
does not have values (empty)	FALSE	TRUE	FALSE	TRUE

Melody™ checks for validity of causality assignments when users attempt to browse SysML instance models in the Melody browser. The causality assignment utility in Melody™ (Melody→Util→ Add default causalities) assigns default causalities based on this table.

6) When populating aggregate properties (see section 6.6.4), Melody™ requires that the names of part / reference / shared properties in the instance model should start with the name of corresponding part / reference / shared properties in the schema model followed by an underscore ("_"). For example, in section 6.6.4, the part properties of the ParkingSpace_at_6AM block instance in Figure 18 are named parkedVehicles_1, _2, and so on. These names start with the name of the corresponding part property parkedVehicles (block ParkingSpace in Figure 17) followed by an underscrore. Note that this is a workaround by Melody™ to model SysML instances (not supported in Rhapsody 7.5.2).

6.2 Naming requirements

1) All SysML block elements should have a unique name, even though they are in different packages.

2) All SysML blocks, constraint blocks, and properties (part / reference / shared / value) should have a name.

3) A block and its parent package cannot have the same name. For example, a block named A cannot be owned by a package named A.

4) A block and a package at the same level cannot have the same name. For example, a block named B and a package named B cannot be at the same level—owned directly by another package.

5) 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.

6) All SysML model elements should have names that start with an alphabetical character (A-Z, a-z).

7) The allowable character classes that can be used in naming model elements are: A-Z, a-z, and 0-9. Underscore ("_") can be used for naming blocks and block properties but not constraint properties.

8) The period/dot operator (.) 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) All value properties that are participating in the parametric model are expected to be populated by real numbers within the bounds[f] specified by Java Double type.

2) All constraint parameters in parametric models must be of value type SysML Real or its subtypes.

3) All numeric values in attributes must begin with a numerical character 0-9. Values beginning with a space or decimal point may not be read correctly.

4) Mathematical functions should use the syntax defined in section 6.4.

5) A valid mathematical equation (for Melody™) is one that has a single variable on the LHS. For example, `a=b+c` will be processed but users may face issues processing the same equation defined as `b+c=a`.

6) The list of math operators supported[g] in Melody are as follows:
   a) Addition (+), Subtraction (-), Multiplication (*), Division (/)
   b) Unary plus (+x), and Unary Minus (-x)

7) It is recommended that large expressions be broken into simpler expressions that are more readily solvable by Mathematica and OpenModelica. 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 Melody™ 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 Melody is based on JEP[h] (with extensions).

6.4 Math constants and functions

Melody™ currently supports the following math constants and functions that may be used in defining constraint specifications.

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.

A list of constants is provided below.

[f] http://java.sun.com/javase/6/docs/api/
[g] Note that Melody™ may not warn if other operators are used. Users should only use the math operators stated here.
[h] JEP: http://www.singularsys.com/jep/

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 Rhapsody 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 Melody browser (see Figure 23 in section 7.2) and press Solve. Note that this selection will not be stored in Melody™ 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:

1) For large expressions involving several functions, Mathematica may not always return an answer when these expressions are evaluated in non-natural directions.
2) The current version of Melody™ 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 Melody™.

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) will return x=n*1.570 (or n*pi/2) where n is a positive integer not necessarily equal to 1.

Name	Syntax	Example
Sine	sin(x)	y = sin(x)
Cosine	cos(x)	y = cos(x)
Tangent	tan(x)	y = tan(x)
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(x1,x2,x3,…) For Mathematica, $x_1,x_2,…$can be arrays or single-valued For OM, $x_1,x_2,…$can only be single-valued	y = min(x1,x2,x3)
Max (returns the argument with maximum value)	max(x1,x2,x3,…) For Mathematica, $x_1,x_2,…$can be arrays or single-valued For OM, $x_1,x_2,…$can only be single-valued	y = max(x1,x2,x3)
Average (returns the arithmetic mean of members of the array passed as argument)	avg(x) where x is an array	y = avg(x)
Sum (returns the sum of the members of the array passed as argument)	sum(x) where x is an array	y = sum(x)

6.5 Conditional Functions and Operators

Melody™ currently supports conditional statements in the following format.

<Result> = if(<Condition>, <Result if Condition is TRUE>, <Result if Condition is 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)

6.6 Aggregate Properties and Functions

Melody™ supports primitive aggregates—value properties that contain a list of one or more values. 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 a SysML model when a value property is a collection (list) of numbers. Melody™ currently supports several types of relations involving primitive aggregates—see below.

6.6.1 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.

6.6.2 Instance Attribute (Slot) Values

When setting up an instance for solution, values are assigned to attributes (slots) for each variable—either Real values for givens or empty values as placeholders for unknowns/targets. The values are specified in the Initial Value field of the attribute. Use commas to separate values. For example, if an attribute has multiplicity 1..* in the schema and has causality "given" for parametric solving purposes, then its values may be specified as 1,2,3,4. If the attribute is a target/undefined, then its values may be specified as spaces separated by commas (e.g. , , , ,). Users must specify the exact number of values (as numbers or spaces) for all attributes in the instance model. A value property must be populated by an array that has all values as givens (e.g. 1,2,3,4) or all as undefined/target (e.g. , , ,) but not a combination (e.g. 1,2, ,4) – see item 8) in section 0.

6.6.3 Aggregate Functions and Operators

The following function and operators are supported by Melody™ 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	x is an aggregate of Real numbers and n is a single Real number;

Not supported for OM	{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.

Equations may involve both single-valued variables/constants and multi-valued (array) variables if Mathematica 7.0 (or higher) is used as the core solver. Examples of such equations are:

1) x = a, where x is an aggregate and a is a single-valued variable
2) x = a + n, where x and a are aggregates and n is a single-valued variable
3) x = a + pi(), where x and a are aggregates
4) x = a + rand(), where x and a are aggregates

If OpenModelica is selected 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.

6.6.4 Complex Aggregate Relationships

Melody™ R3 can execute complex aggregate relationships. A ***complex aggregate relationship*** is a SysML parametric relation (constraint property) that involves value properties owned by *aggregate* part/reference/shared properties. The term *aggregate* implies that the upper bound of multiplicity > 1, for e.g. multiplicity = 1..* or 0..*. **Complex aggregate relationships allow users to define parametric models independent of the number of complex items (blocks) in a collection. The number of items is specified in the instance model. Melody™ executes the complex aggregate relationship for an instance model.**

Figure 16 below illustrates a Parking Space system composed of 1 or more parked vehicles (e.g. cars, trucks, carts). Figure 17 is a parametric diagram of the ParkingSpace block. The spaceCalc constraint property is a complex aggregate relationship since it involves a value property (parkingArea) that is owned by a collection—part property parkedVehicles has multiplicity 1..*. The spaceCalc constraint property is used to compute the parking space occupied by all vehicles by summing the parkingArea of n vehicles (where n is undefined at this point). The pcCalc constraint property is used to compute the occupationRatio (ratio of occupied space to the available space).

Figure 18 illustrates an instance of the ParkingSpace system and represents the parking space at 6 AM. The parking space has 2 cars, 3 trucks, and 2 carts. Executing the parametric model Figure 17 for this instance shows that 24% of the parking space is occupied at 6 AM. Figure 19 illustrates another instance of the ParkingSpace system and represents the parking space at 9 AM. The parking space has 4 cars, 5 trucks, and 2 carts. Executing the parametric model Figure 17 for this instance shows that 41% of the parking space is occupied at 9 AM.

Melody™ requires that the names of part / reference / shared properties in the instance model should start with the name of corresponding part / reference / shared properties in the schema model followed by an underscore ("_"). For example, the part properties of the ParkingSpace_at_6AM block instance in Figure 18 are named parkedVehicles_1, _2, and so on. These names start with the name of the corresponding part property ParkingSpace.parkedVehicles in the block structure shown in Figure 17.

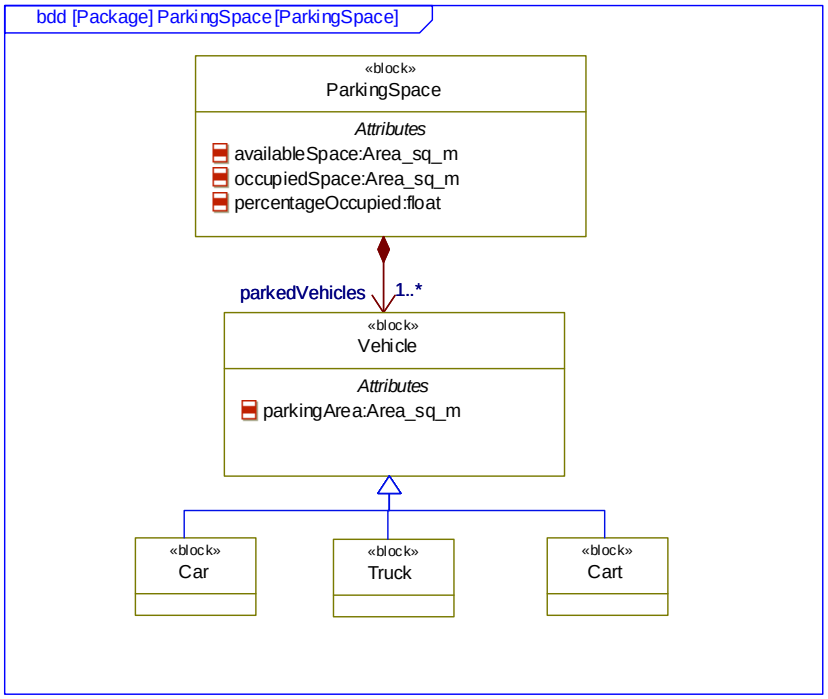


Figure 16: Parking Space model with aggregate part property (ParkingSpace.parkedVehicles)

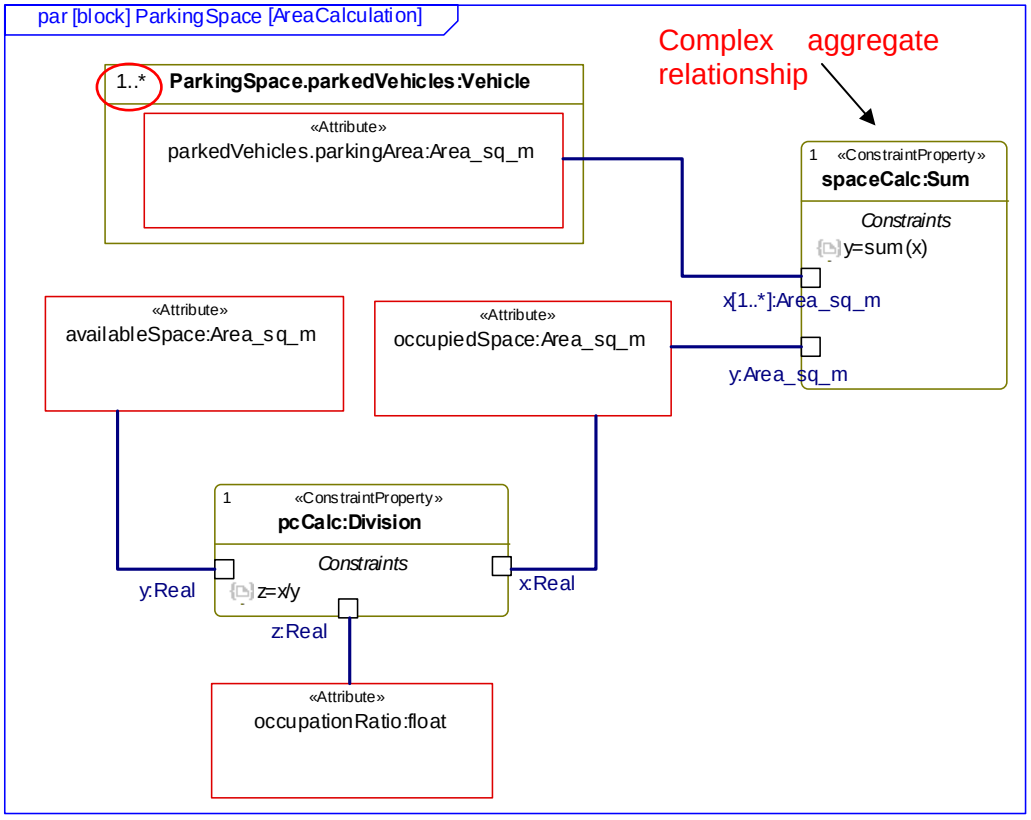


Figure 17: Parametric diagram illustrating a complex aggregate relationship

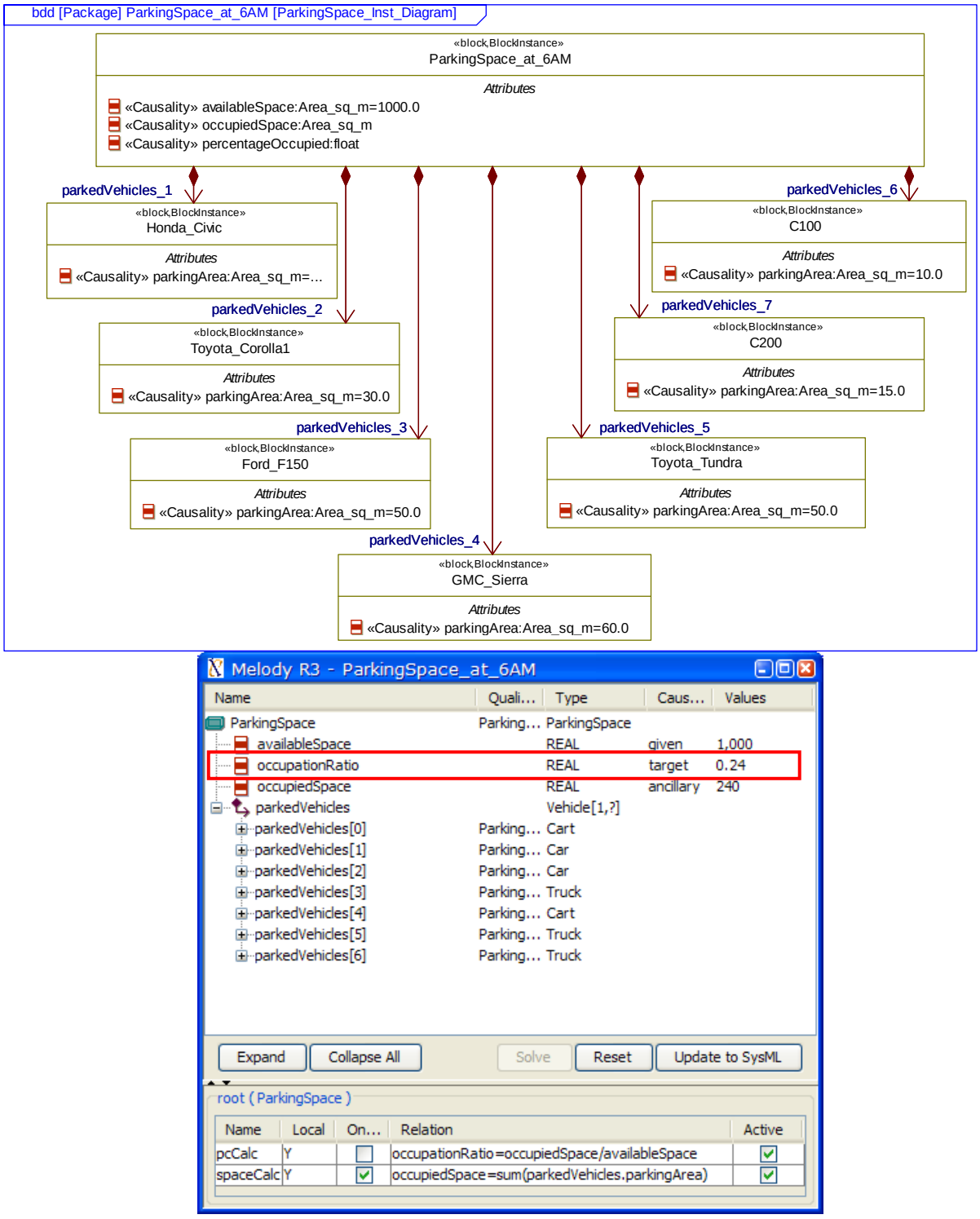


Figure 18: Results of executing parametric model for parking space with 7 vehicles — 2 cars, 3 trucks, and 2 carts

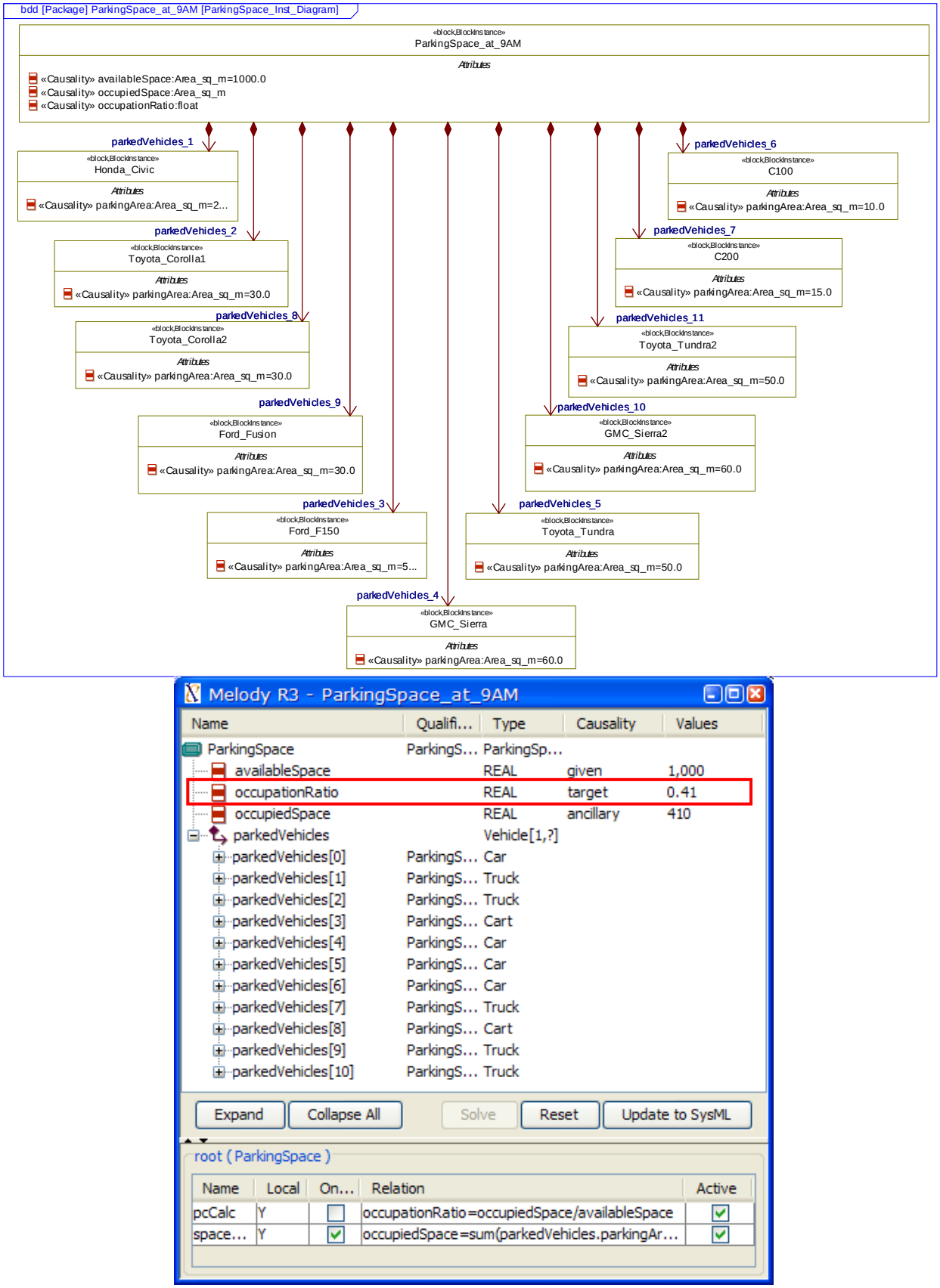


Figure 19: Results of executing parametric model for parking space with 11 vehicles — 4 cars, 5 trucks, and 2 carts

6.7 Limitations

Melody™ R3 has the following limitations with respect to SysML models.

1) OpenModelica (OM) as the core solver — If OM is selected as the core solver, some math functions are not supported or have a limited support. These functions are commented in blue in sections 6.4, 6.5, and 6.6.

2) Constraint block-related limitations - This version of Melody™ does not support:
   a) multiple constraint specifications in a constraint block.
   b) constraint blocks composed of other constraint blocks.
   c) binding connectors between constraint parameters belonging to two different constraint properties. A user must create an additional value property to achieve this.
   d) inequality constraints.

3) Instance-related limitations – Since Rhapsody 7.5.2 does not support SysML instances (InstanceSpecification), instances are modeled as blocks with stereotype <BlockInstance> for Melody™ to solve parametric models and perform trade studies - see section 6.1.2 for details. Though an instance is cognizant of the corresponding block, it is not synchronized with the features of that block. If a user adds/removes block properties, or changes their names, the instance block will not be automatically updated. This update has to be performed manually by the users. Melody™ provides automated instance generation capability to avoid potential mistakes in manually creating instances.

4) Value type-related limitations – Attributes with string values must be typed by the String value type available with the Melody™ Profile. All value properties, except for those typed by the String value type, will be expected by Melody™ to be populated with real numbers irrespective of the specific value type used for typing these value properties. Ideally, Melody™ would require users to model their custom value types as specializations of the Real value type available with the SysML Profile in Rhapsody. Since value types cannot be specialized in Rhapsody 7.5.2, this restriction has been relaxed for Melody™ users.

5) Cyclic references-related limitations – This version of Melody does not support some types of cyclic references among model schema elements, and among model instance elements. Specifically, the following scenarios are not supported:
   a) A block has a property that is typed by the block itself.
   b) Given two blocks, each block has a property typed by the other block (e.g., A.a1 is of type block B, and B.b1 is of type block A).

6) Units – Melody™ does not support automated unit conversions in math expressions but *it provides a validation utility that warns users if the units of value properties or constraint parameters connected using binding connectors are incompatible*. This validation utility is invoked when a user attempts to validate a schema/instance model, or browse/solve an instance model.

7) Complex numbers – This version of Melody™ does not support complex numbers. If solution results return a complex number for a variable, the string "No Value" is displayed for that variable in the Melody™ browser.

8) If a value property is populated with an array in the SysML instance model, all values of the array should either be givens (e.g. 1,2,3,4) or undefined/targets (e.g. , , ,) but not a combination (e.g. 1,2, ,4). This restriction is primarily due to lack of support for SysML instances and slot values in Rhapsody 7.5 and 7.5.1.

9) This version of Melody™ does not support scientific notation for numbers.

10) If you change the causality of a solved target variable to given in Rhapsody and launch the Melody™ browser, you may see an over-constrained set of values. Melody 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.

11) When you successfully validate the schema/instance structure of a model (see Validate in section 7.1), you will receive the message in Figure 20/Figure 21.
    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 Melody™ does not check for over-constrained equations and similar non-syntactic issues. In general the math solver (invoked via the Melody™ Browser) will uncover such issues usually by returning a "No Value" result or an over-constraint warning.

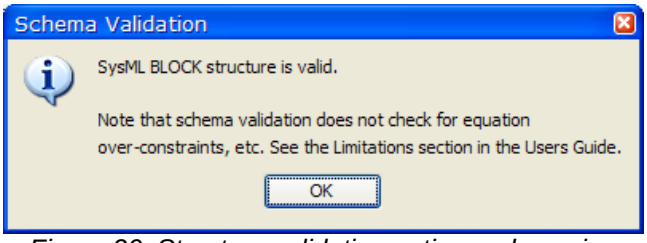


Figure 20: Structure validation notice and warning

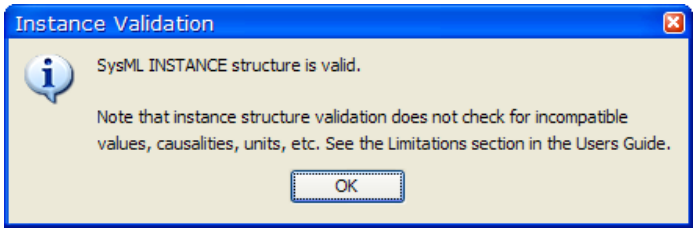


Figure 21: Instance validation notice and warning

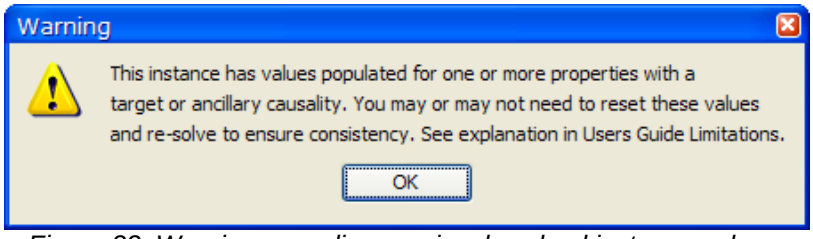


Figure 22: Warning regarding previously solved instance values.

12) When you open an existing instance model in Rhapsody for browsing (see Browse in section 7.1) you will see the warning message in Figure 22 if that instance model has values populated for any attributes with causality "target" or "ancillary". 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*.

13) There are several ways that previously solved instance values can become invalid, because the instance in Rhapsody is not continuously connected to the instance in the Melody™ browser), including:
    a) Someone changed the corresponding schema structure in Rhapsody after that instance was solved (e.g., they added a new equation or changed an existing equation).
    b) Someone changed an attribute value or causality in the instance in Rhapsody. In general we recommend not making such changes in Rhapsody for a previously solved instance. Instead, open the Melody™ browser, make such changes, and then update the SysML model.

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 Melody™. With the changes in the model, the values in old instances may be non-conformant to the model schema. This version of Melody™ 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 Melody™ browser, (c) Reset all non-given values (automatically by pressing the Reset button), and (d) solve the instance again.

14) Mathematica solving issues:
   a) If a system of equations is under-constrained, Melody™ 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 (*melody@intercax.com*) for further information.

7 PROGRAM FEATURES

7.1 Command Menus

Melody™ commands are applied by right-mouse-clicking on the appropriate block or package in the Rhapsody model tree and selecting the menu item Melody. The list of commands and their description is below.

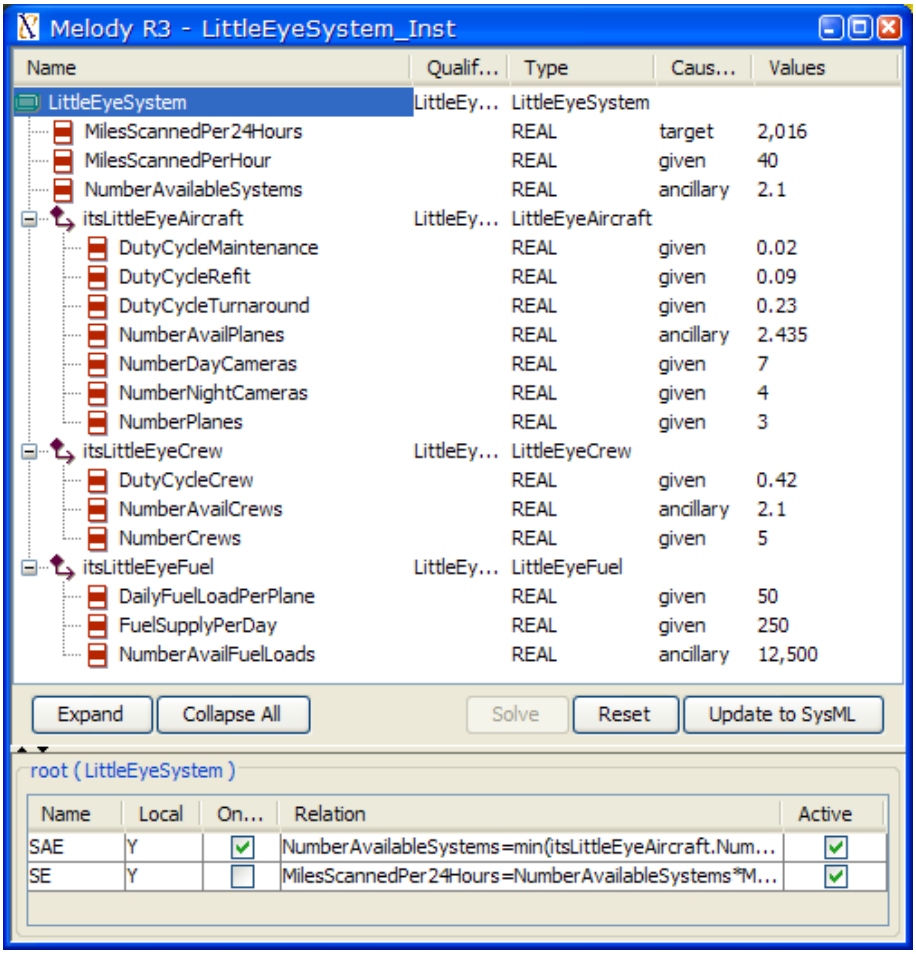
Command	Description
Validate	• Validates the block structure (schema model) when invoked on a block. This command validates the specific block on which it is invoked, and recursively validates: (1) all blocks related as part, reference, and shared properties, and (2) all constraint blocks related as constraint properties. • Validates the block instance structure (instance model) when invoked on a block instance. This command validates: (1) the block corresponding to the instance, and all related blocks in a recursive manner (as above), and (2) the block instance and all related instances in a recursive manner.
Browse	Launches the Melody™ browser (see section 6.2) to browse and solve the block instance (and related instances) on which it is invoked. The Browse command performs several functions when invoked on an instance model. • Validates the schema model for the instance model (as done by the Validate command) • Validate the instance model (as done by the Validate command) • Launches the Melody™ browser for the instance model
Util → Create Instance	Creates an instance structure and diagram (BDD) for a given block. When invoked on a block, this command instantiates the given block and all its part /

	reference / shared properties recursively to create an instance structure. It also creates a block definition diagram showing the instance structure (newly created instances and relationships among them).
Util → Assign default causalities	Assigns default causality to instance slots (attributes of block instances). Attributes that have been assigned a value are tagged as given, and attributes without values are tagged as undefined. At least one variable must be assigned manually as target to solve the model.
Util → Select Instantiated Block	Selects the classifier block when invoked on a block instance. The classifier block is the block that was instantiated to create the block instance.
Util → Copy Element GUID	Copies the GUID (unique identifier of the element in Rhapsody) to the clipboard and also shows it in the log window, when invoked on any element in the model tree
Util → Refactor Binding Connectors	• Moves all binding connectors related to a block from their current location in the model to inside the block, when invoked on a block. • Performs the above for all blocks in a package when invoked on a package.
Util → Add Instances to Aggregate Property	Adds instances to populate an aggregate part, reference, or shared property when invoked on the property.
Excel → Setup	Launches the Excel setup utility to connect block/instance attributes to Excel spreadsheets. See Table 2 for details.
Excel → Read from Excel	Reads values from Excel spreadsheet(s) into the SysML model in Rhapsody. See Table 2 for details.
Excel → Write to Excel	Writes values from SysML model in Rhapsody to Excel spreadsheet(s). See Table 2 for details.
Trade Study → Setup	Launches the trade study setup utility for specifying number of scenarios.
Trade Study → Run	Runs a trade study.
Help → About Melody	Provides information about your Melody™ installation.
Help → Users Guide	Launches the Melody™ Users Guide which is located in the doc folder under

	your Melody™ installation, for e.g. C:\Program Files\IBM\Rational\Rhapsody\7.5.2\Share\Profiles\Melody\doc\Users_Guide.pdf
Help → Tutorials	Launches the Melody™ Tutorials document which is located in the models\tutorials folder under your Melody™ installation, for e.g. C:\Program Files\IBM\Rational\Rhapsody\7.5.2\Share\Profiles\Melody\models\tutorials\Tutorials.pdf

7.2 Browser

The Melody™ 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 23 below.



Variable Browser (shown in expanded form)

Toolbar

Relationship Browser

Figure 23: Melody™ Browser

7.2.1 "Solution in progress" Window

When the Solve button on the browser is clicked, Melody™ orchestrates the solution of the network of parametric equations using solvers (such as Mathematica, OpenModelica, and MATLAB). During this interval, a Solution in progress window is displayed, as shown in Figure 24 and Figure 25 below.

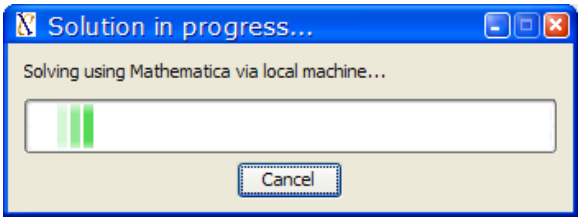


Figure 24: Solution in progress window when using local Mathematica

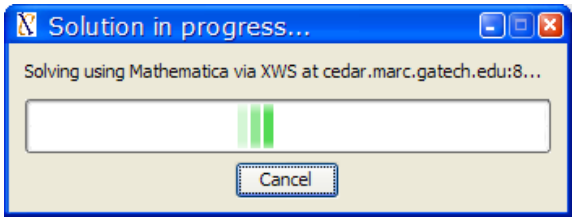


Figure 25: Solution in progress window when using remote Mathematica

The Cancel button causes Melody™ to ignore any intermediate results returned from Mathematica, 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 (e.g. Mathematica) directly, i.e., the solver job may continue to execute, in which case Melody™ will ignore any results it obtains.

If the solver job itself hangs, it may require manual intervention before you can continue successfully solving models via Melody™. To cancel a solver job, consult your solver's Users Guide (for local solver installation) or contact your server administrator (for an XWS/web services-based solver installation).

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

The commands available via the Toolbar are described below.

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/OpenModelica 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 Rhapsody

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. itsLittleEyeAircraft in Figure 23, 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/OpenModelica. 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 Rhapsody and then re-open in the Melody™ 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.

7.3 Melody.ini file

The Melody.ini file provides settings to control Melody™ behavior. The file is located here: <Rhapsody_Root>\Share\Profiles\Melody\xfw\conf.

The role of each variable and the format for specifying its values are described below. If there are restrictions on the values allowed for a variable, these are specified in the following format: Variable name = allowable value 1 / value 2 / …

1) com.intercax.xaitools.solver.name = Mathematica / OpenModelica

   This variable is used to specify if Melody should use Mathematica or OpenModelica as the core solver. Allowable values for this variable are Mathematica or OpenModelica.

2) com.intercax.xaitools.local.mathematica.true.or.false=true / false

   This variable is used to specify if Melody should use local or remote Mathematica. Set value = true for local Mathematica, or false for remote Mathematica (via XWS). If the value is set to false, ensure that you specify the location of remote Mathematica (com.intercax.xaitools.soap.mathematica.serverhost) and the access key (com.intercax.xaitools.soap.mathematica.accesskey). Contact your XWS administrator for access to remote Mathematica. For information on remote Mathematica (via XWS), refer to Step 5 in section 4.2.

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

   This variable is used to specify the time interval in seconds—measured after pressing the Solve button in the Melody browser—after which Melody will disconnect from the external solver (e.g. Mathematica or MATLAB). Note that this will not kill the solver runtime process. However, Melody will not wait for the solution results. By default, the value of this variable is set to 180 seconds (3 mins). If you estimate your problem will take longer to solve, increase the value of this variable accordingly. If you do not want Melody to timeout, specify a value less than 0. For example, setting the value as -1 will ensure that the solution process is not timed out.

4) com.intercax.xaitools.soap.mathematica.serverhost = IP address:port number

   This variable is used to specify the location of remote Mathematica installation. Values are specified in the following format: IP number (or alias): port number. For information on remote Mathematica (via XWS), refer to Step 5 (Option 3) in section 4.2.

5) com.intercax.xaitools.soap.mathematica.accesskey = someKey

   This variable is used to specify the access key for the remote Mathematica installation. For information on remote Mathematica (via XWS), refer to Step 5 (Option 3) in section 4.2.

6) com.intercax.xaitools.temp.dir= temp

   This variable is used to specify the location of temp folder where the Mathematica job is created and interim results are stored during the solution process. The location is specified relative to the xfw folder — <Rhapsody_Root>\plugins Share\Profiles\Melody\xfw.

7) RetryInvervalInMilliSeconds = 1000

   When using remote Mathematica (via XWS) network issues or other problems may hinder Melody™ from connecting to Mathematica. This variable is used to specify the time interval (in milliseconds) after which Melody™ will retry connecting to Mathematica.

8) NumberOfRetry=1
When using remote Mathematica (via XWS) network issues or other problems may hinder Melody from connecting to Mathematica. This variable is used to specify the number of times Melody™ should retry connecting to Mathematica.

9) xws.urn.version = urn:XWS_v2.2

This variable is used to specify the XWS service used for connecting to remote Mathematica. For information on remote Mathematica (via XWS), refer to Step 5 (Option 3) in section 4.2.

10) maxNumberOfDecimalsToDisplay = -1

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

8 CONNECTIONS TO EXTERNAL TOOLS

8.1 Melody™ - Excel Connection

The Melody™ - Excel Connection (MEC) allows users to read/write slot values from/to Microsoft Excel files. This enables users to populate instances values from Excel workbooks[i] before solving the model in Melody™, and export solved instance values to Excel workbooks. Hence, Excel spreadsheet-based templates can be used with Melody™ in a sequential manner. For example, users may compute some parameters using spreadsheets, read these values in Rhapsody to populate some SysML value properties (attribute), solve the SysML model using Melody™, and export solved values to spreadsheets.

The Satellite tutorial provided with Melody™ is an example usage of the MEC feature. It is located under <Rhapsody_Root>\Share\Profiles\Melody\models\tutorials\Satellite.

MEC can be invoked on both blocks and instances—blocks with stereotype BlockInstance—in Rhapsody. This allows users to take advantage of MEC's capabilities even if they are not using Melody's parametric solving capabilities.

The operations, features, and limitations of MEC are described below in sections 8.1.1, 8.1.2, and 0 respectively.

Note: It is recommended that you first try this feature (esp. Excel write operation) on a backup copy of your spreadsheet before trying it on a spreadsheet for production usage. This will ensure that you are aware of the updates that the feature performs on your spreadsheet.

[i] The terms Excel file and Excel workbook mean the same.

8.1.1 Operation

Follow the steps below to use MEC to connect your SysML models to Excel spreadsheets.

Step 1. *Setup attributes to interact with Excel worksheet*

To setup attributes to interact with Excel, right click on the attribute (or its parent block) in Rhapsody's model tree and select Melody→Excel→Setup, as shown in Figure 26 below. Once the Setup menu is selected, Melody Excel Setup utility appears, as shown in Figure 27 below.

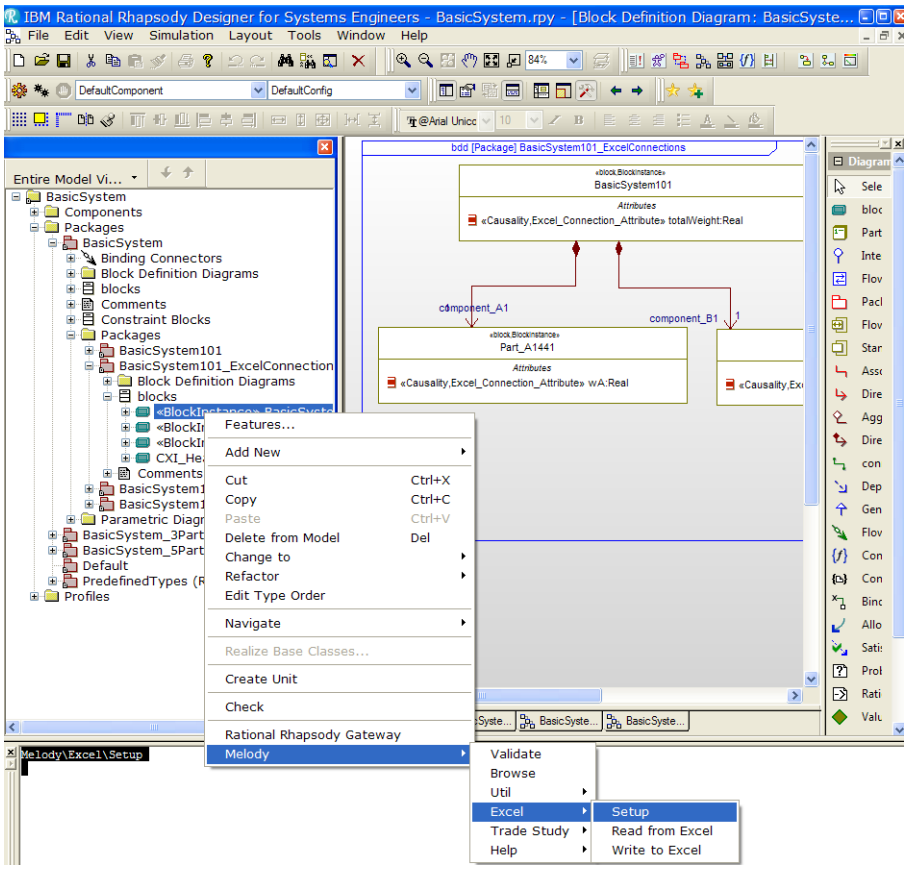


Figure 26: Setup Excel connection for attributes

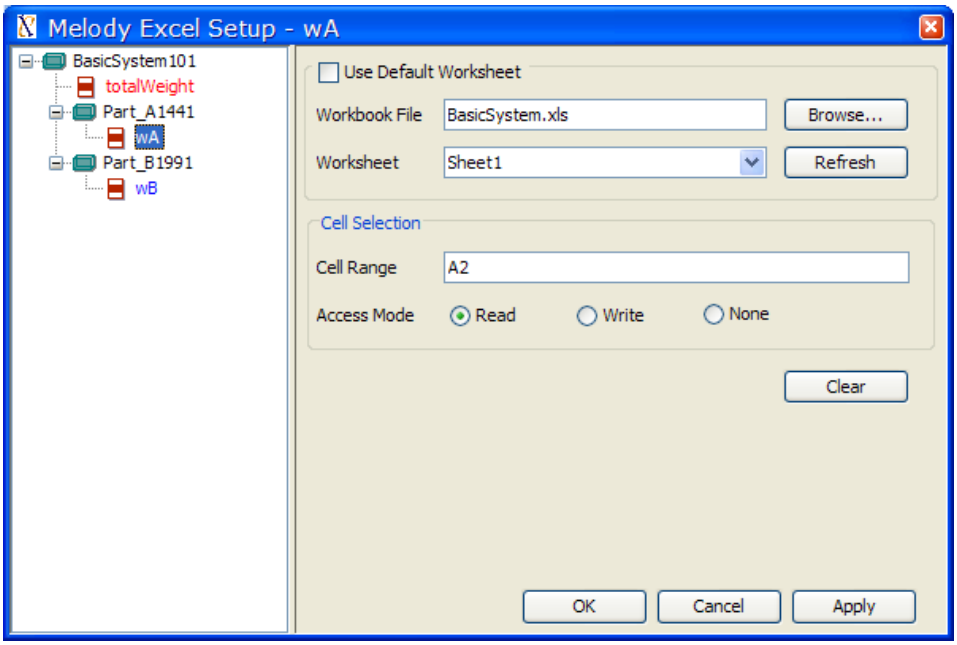


Figure 27: Melody Excel Setup utility

The setup utility shows the entire block structure (part/reference/shared properties) for the selected block (or the block owning the selected attribute). If the setup utility was invoked on an attribute, then that attribute is shown highlighted in the model tree shown in the setup window. Users can now click on specific attributes and link them to Excel spreadsheets. If an attribute has already been connected to Excel, the connection information will be shown in the setup utility.

The setup utility can also be invoked on packages. See section 8.1.2 for details.

Step 2. *Specify Excel connection for attributes*

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

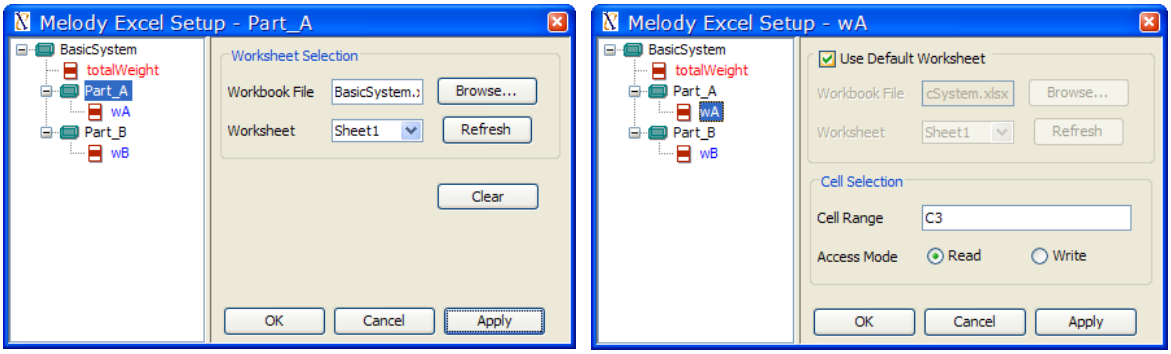
a) Select the attribute in the setup window (as shown in Figure 27).

b) Specify the location of the Excel workbook associated with the attribute in the *Workbook File* field. If the workbook file is located in the same folder as the Rhapsody model file (.rpy), 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 Rhapsody file enhances the portability of the model. The Rhapsody model file and the workbook files can be distributed together without system-specific folder location settings.

c) Select the worksheet associated with the attribute. 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 attributes of a block are to be linked to the same workbook and worksheet, it is preferable to specify workbook and worksheet information at the block level and then

select the Use Default Worksheet option for all such attributes. Figure 28 below shows an example where the Excel workbook and worksheet is specified for the block Part_A. The attribute wA (owned by this block) uses the worksheet.



Excel worksheet specified for the instance Slot uses the default worksheet

Figure 28: Specifying workbook and worksheet at the instance level and using it for slots

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, cell range C6:C15 in Sheet1 is named as PowerUsage. The scope of this name is the worksheet Sheet1. Figure 29 below shows an Excel dialog to name cell ranges.

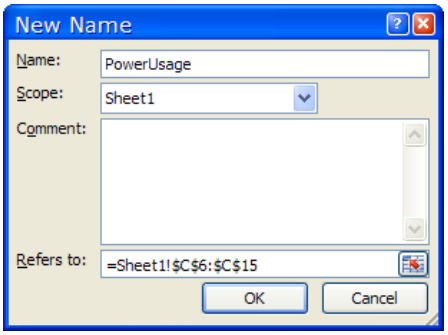


Figure 29: Naming cell ranges in Excel

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 attributes: <first cell address>:<last cell address>. For example, if a multi-valued attribute is to be associated with cells from A2 to A10, cell range should be specified as A2:A10.
   (2) for single-valued attributes: <cell address>. For example, if a single-valued attribute 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 an attribute to an Excel spreadsheet using MEC, 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) MEC 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 27, attributes that are setup to read from spreadsheets are shown in blue and attributes that are setup to write to spreadsheets are shown in red.

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

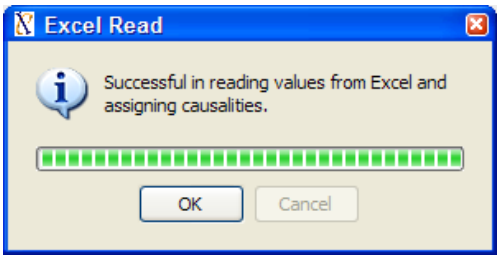
The table below summarizes the commands that can be issued from the MEC setup window.

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 Rhapsody model
OK	Saves the Excel connection settings to the Rhapsody model (Apply) and closes the setup utility
Clear	Removes the Excel connection settings for the subject slot
Cancel	Closes the MEC setup utility without saving the Excel connection settings

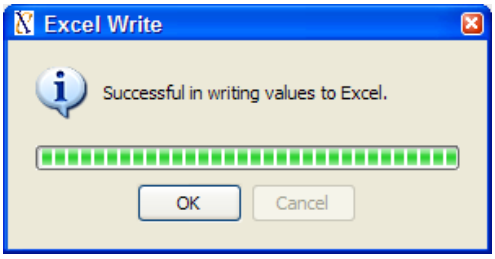
Step 3. *Execute Excel Read / Write operation on attributes*

Once an attribute is connected to Excel, read/write operations can be executed by right clicking the attribute and selecting Melody→Excel→Read from Excel or Write to Excel menus. When the read/write operation is executed successfully, corresponding information messages pop-up as shown in Figure 30.

During the Excel Read operation, the causality of all instance model attributes read from Excel is automatically changed to "given" (from "undefined") in accordance with Table 1. Once an attribute 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.



MEC message for successfully reading attribute values from Excel



MEC message for successfully writing attribute values to Excel

Figure 30: Information messages for successful execution of MEC read/write operations

Steps 1-4 above demonstrate the operation of MEC in Melody™ for attributes. Additional steps for more efficient operation of MEC are described below.

Step 4. *Execute Excel Read/Write operation for a block/ instance or a schema/instance package*

Excel read/write operations can be invoked for an attribute (as demonstrated above), a block/instance, or a schema/instance package. To invoke Excel Read/Write operation on an block/instance (or block/instance package), right click the block/instance (or block/instance package) and select Melody→Excel→Read from Excel or Write to Excel menu.

If the Excel Read operation is invoked on a block/instance, MEC will execute the Excel Read operation for all attributes of that block/instance that are setup to read from Excel (Access Mode=Read). Similarly, if the Excel Write operation is invoked on a block/instance, MEC 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 block/instance package, MEC will perform the Excel Read/Write operation for all blocks/instances in the instance package.

Step 5. *Initializing slots with empty values before solving with Melody.*

If a slot value is intended to be a target or an undefined variable for Melody 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 PM-EC, 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 Melody browser.

The table below summarizes the PM-EC commands, the arguments on which these commands may be issued, their behavior, and the response message on successful execution of commands. These commands are available under Melody→Excel menu.

Table 2: Commands for Melody™ - Excel Connection

Command	Arguments (Invoked on)	Description	Messages
Setup	Attribute	Opens the Excel Setup utility for the block owning the attribute, and the subject attribute is shown highlighted. If the block is a part of a system model (schema), then the setup utility is launched for the root block and the subject attribute is shown highlighted.	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.
	Block or Instance	Opens the Excel Setup utility for the block/instance. If the block is a part of a system model (schema), then the setup utility is launched for the root block and the subject block is shown highlighted.	
	Package	Opens the Excel Setup utility for the root block or root instance in the package.	
Read	Attribute	Reads values from an Excel spreadsheet and populates attribute values if Excel access mode=Read. If the attribute is owned by an instance, the causality is set to "given" after reading values from Excel.	• If attribute 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 attribute 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 attribute values are not read from Excel, response messages (indicating the problems) are shown in the Rhapsody Log window.
	Block/Instance	Executes the Excel Read operation for all attributes directly owned by the block.	A successful response message is show only when the Excel read and causality

		If invoked on an instance, the causalities of all attributes, whose values are read from Excel, are set to "given".	assignment operation is successful for all attributes (with read access mode) in the block/instance. Else, no response message is shown.
	Package	Executes the Excel Read operation for all attributes of all blocks/instances in the package. If invoked on an instance package, the causalities of all attributes (of all instances in the package), whose values are read from Excel, are set to “given”.	A successful response message is shown only when the Excel read and causality assignment operation is successful for all attributes of all blocks/instances in the instance package. Else, no response message is shown.
Write	Attribute	Writes attribute values to an Excel spreadsheet if Excel access mode = Write for that attribute.	• If attribute values are successfully written to Excel, the response message states "Successful in writing values to Excel". • If attribute values are not written to Excel, response messages indicate the problem.
	Block or Instance	Executes the Excel Write operation for all attributes directly owned by the block.	A successful response message is shown if the Excel Write operation is successful for all attributes of the block/instance.
	Package	Executes the Excel Write operation for all attributes of all blocks/instances in the package.	A successful response message is shown only if the Excel Write operation is successful for all attributes of all blocks/instances in the package.

8.1.2 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) Excel Read/Write operations can be invoked for an attribute, a block/instance, or a package containing blocks/instances.

3) Excel Read/Write operations can be performed for both numerical and string values. If a value property (attribute) connected to Excel is typed by the String value type (available in the Melody™ Profile), text-based values can be read from a spreadsheet to that attribute and vice versa. However, if a value property is typed by any other value type (not String), only numerical values can be read successfully from a spreadsheet to that attribute and vice versa. In this case, non-numeric values read from Excel will be shown as null in the Initial Value field of that attribute.

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

8.1.3 Limitations

1) The Excel workbook needs to be closed before the Excel Write operation is invoked on an attribute or block/instance or package.
2) The Excel Write operation will not create a new workbook file if it does not exist.
3) MEC 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 MEC 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, users are recommended to first use MEC on a copy of their original workbooks and test if the MEC Read/Write operations preserve the specific features in their workbooks.

8.2 Melody™ - Custom Mathematica Connection (available only with Melody™ Pro)

The Melody - Custom Mathematica connection (a.k.a cMathematica) exposes the full power of Mathematica to Melody™ 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 Melody™. Users can create as many custom functions as desired if they have familiarity with Mathematica function programming and access to the Mathematica directory structure. The canonical form of the constraint expression is as below and it would be used in a parametric diagram as shown in Figure 31

output parameter = cMathematica(function name, input arguments)

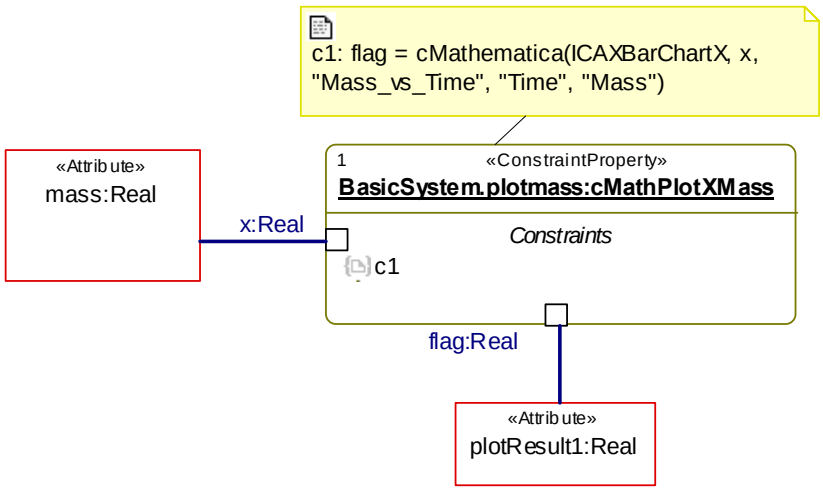


Figure 31: Constraint expression using cMathematica function in a parametric diagram

The figure above shows a usage of the constraint block (cMathPlotXMass) as a constraint property (plotmass) of the block BasicSystem. The constraint c1 of the constraint block cMathPlotXMass has the following specification in the Rhapsody SysML model.

flag = cMathematica(ICAXBarChartX, x, "Mass_vs_Time", "Time", "Mass")

For illustration purposes, this is also shown as a note anchored to the constraint property in the parametric diagram.

A template constraint block for cMathematica functions is available with Melody's Constraint Block Library as below.
Melody::Constraint_Block_Library::External_Functions::Mathematica::cMathematica_Function

The BasicSystem_cMathematica example model in your Melody™ installation demonstrates this feature. This model is included in the models\other_examples folder of your Melody™ installation, for e.g.
C:\Program Files\IBM\Rational\Rhapsody\7.5.2\Share\Profiles\Melody\models\other_examples

This model is described in the Other_Examples.pdf document which is located in the same folder.

8.2.1 Installation

A library of pre-defined Mathematica graphing and statistical functions is provided with Melody™. After Melody™ installation, this library is located here: <Rhapsody_Root>\Share\Profiles\Melody\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 Melody™ using our test server, then you do not need to follow this step.

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.

8.2.2 Usage

Tutorial 4 (LittleEye), Step III (Item 5-6) describes the process of creating a constraint block and specifying constraints that wrap MATLAB M-files. The process is same for custom Mathematica functions except for the syntax of the constraint expressions. The constraint expression should be in the following form (as also illustrated in Figure 31 above) where function name is the name of the custom Mathematica function.

output parameter = cMathematica(function name, input arguments)

Additionally, users can specify folders for exporting plots and other outputs generated by custom-defined Mathematica functions. To achieve this, double click on the constraint in the constraint block (in Rhapsody browser) and apply the External_Model stereotype as shown below. Then, specify the location in the working_dir tag of the constraint.

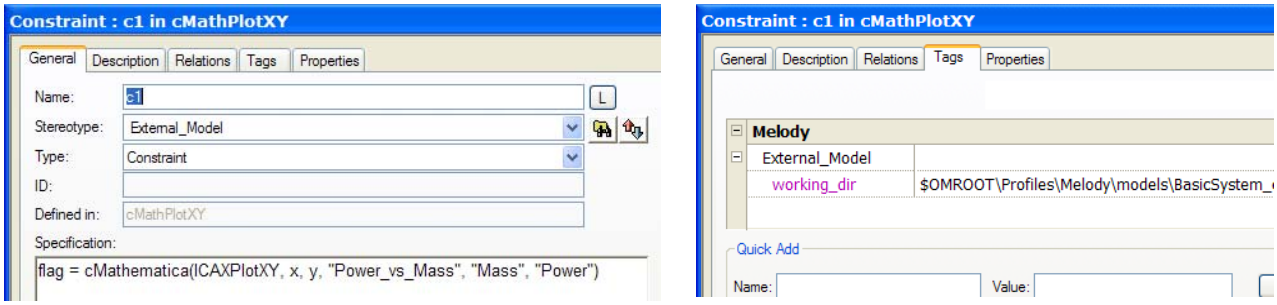


Figure 32: Specifying folder location to export plots and result files from custom Mathematica functions

Next, users must ensure that the first argument in the Mathematica function (in the .m file), is the folder location. The other arguments in the Mathematica function (in the .m file) are the same and in the order specified with the constraint expression.

Note that folder locations are not explicitly specified as an input argument in the expression output parameter = cMathematica(function name, input arguments)

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

2) If the tag working_dir is not populated, Melody™ 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.

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).

8.2.3 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 functions are located in the ICAX.zip folder, as discussed in the beginning of section 8.2.1 above.

These nine functions are shown in the following table, and have several common features (as below).

- 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 Melody™ 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

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”)

8.2.4 Statistical Functions

Three pre-defined statistical functions have been created to provide quick access to some of the statistical power of Mathematica. These functions are located in the ICAX.zip folder, as discussed in the beginning of section 8.2.1 above.

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)

8.2.5 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 Melody™. These five present functions are located in the ICAX.zip folder, as discussed in the beginning of section 8.2.1 above.

8.2.6 UserfnN.m

Within the ICAX library with the pre-defined graphing and statistical functions, we have provided five "empty" functions (as above). 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).

8.2.7 Custom Functions

User can write their own Mathematica functions using the cMathematica capability. We recommend using one of the pre-defined functions as a template to insure compatibility with Melody™.

In order for a function to be recognized by Mathematica, it must be auto-loaded 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.

The Melody™ - Custom Mathematica connection feature is available only with Melody™ Pro edition. For this feature to work, Mathematica (local or remote) must be selected as the core solver. If the Mathematica functions generate output files (e.g. plots) and the user wants to view them on his/her local machine after execution, only local Mathematica should be used. Melody™ does not export output files generated on the remote machine to the local machine. This feature is not supported if OpenModelica is selected as the core solver.

8.3 Melody™ - MATLAB/Simulink Connection (available only with Melody™ Pro)

The Melody™ – MATLAB/Simulink connection (MMC) enables users to wrap MATLAB functions and scripts as SysML constraint blocks and use them in parametric models as regular constraint properties. Melody™ solves for the constraints by invoking MATLAB functions/scripts when required. MATLAB scripts are commonly used to invoke and execute Simulink models. Since MMC 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 following models in your Melody™ installation demonstrate this feature.

1. **LittleEye** example is described in Melody™ Tutorial document (section 5.2, Step III) and included here: <Rhapsody_Root>\Share\Profiles\Melody\models\tutorials\LittleEye. To solve this

model, right click on the Instance01 sub-package under the LittleEye package and select Melody > Browse. Press the Solve button in the Melody browser. After solution, the value of the variable LittleEyeSystem.MilesScannedPer24Hours variables should be 2,016.

2. **Energy** example is available here: <Rhapsody_Root>\Share\Profiles\Melody\models\other_examples\. See the Other_Examples.pdf document which is located in the same folder for a description of the model and instructions to run the model.

3. **BasicSystem_MATLAB** example is available here: <Rhapsody_Root>\Share\Profiles\Melody\models\other_examples\. See the Other_Examples.pdf document which is located in the same folder for a description of the model and instructions to run the model.

A MATLAB script[j] 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[j] accepts input arguments and may have outputs.

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 Melody™ does not require a core solver (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, Melody™ can invoke MATLAB scripts/functions when the constraints need to be solved. Step 1 and Step 2 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 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 Melody™.

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

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

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), Melody™ 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) Open Melody.ini file located under <Rhapsody_Root>\Share\Profiles\Melody\xfw\conf Specify location of the folder as the value of the following variable in Melody.ini file
   com.intercax.xaitools.local.matlab.mfile.location

   For example, if the folder is C:\Data\My_MATLAB_Files, then the variable-value entry in the Melody.ini file should look like:
   com.intercax.xaitools.local.matlab.mfile.location=C:\\Data\\My_MATLAB_Files

Note: *The location of a MATLAB M-file can be specified with each constraint block that wraps the M-file. See Step 5 below.*

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

The Melody.ini file includes a variable that specifies the timeout interval for Melody™ 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 Melody™. Once the timeout is reached, Melody™ 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 Melody Profile. See the following package:
   Melody::Constraint_Block_Library::External_Functions::MATLAB_Simulink

   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). The figure below shows the xfwExternal_MATLAB_Script in the browser and feature windows.

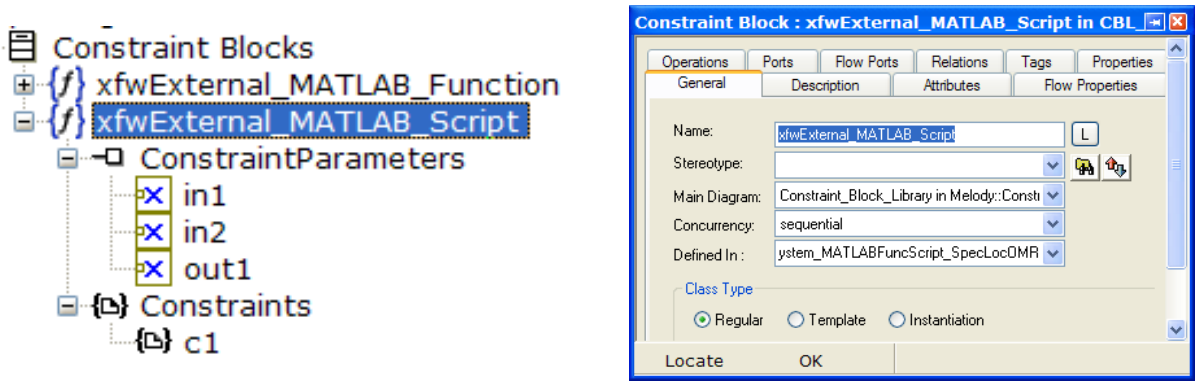


Figure 33: Constraint block for wrapping MATLAB scripts - Browser view (left) and Feature view (right)

3) By default, these constraint blocks are 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.

4) Double click on the constraint c1 in the browser 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, scriptascii,<name_of_ M-file>, <in_param_1>,...) for xfwExternal_MATLAB_Script constraint block, and

   <out_param> = xfwExternal(matlab, function,<name_of_ M-file>, <in_param_1>,...) for xfwExternal_MATLAB_Function constraint block.

   where:
   - <out_param> is the name of the output parameter (result computed during execution and to be read back into SysML instance)
   - <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.

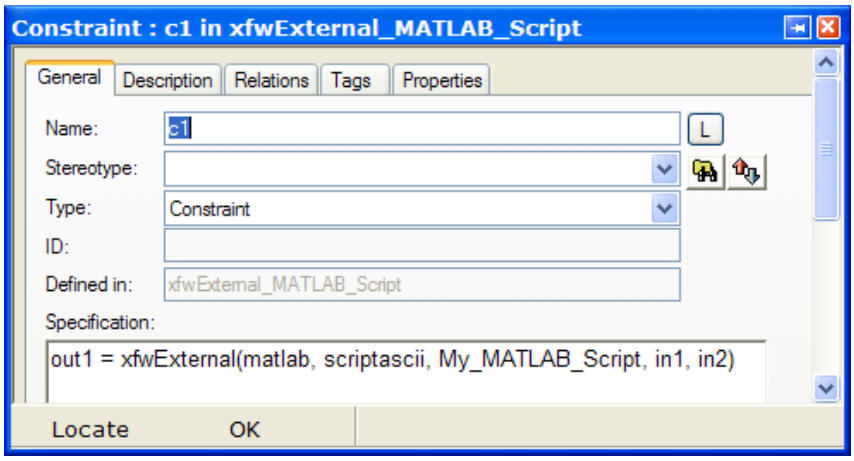


Figure 34: Specification of the constraint in the Constraint Block

In Melody™, input and output parameters of a 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) Apply the stereotype External_Model to the constraint, as shown below.

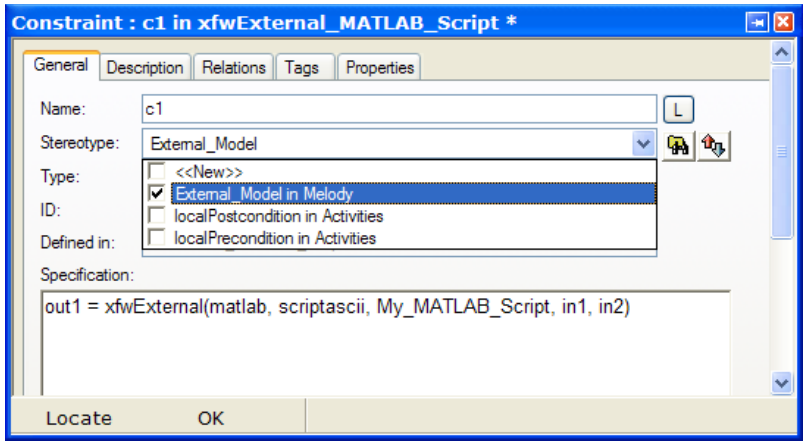


Figure 35: Invoking the constraint specification window from the constraint view

2) Click on the Tags tab and populate the working_dir tag with the location of the folder containing the MATLAB M-file. You can specify an absolute path or a relative path as shown in the figure below. When specifying a relative path, $OMROOT resolves to <Rhapsody_Root>\Share, e.g. C:\Program Files\IBM\Rational\Rhapsody\7.5.1\Share

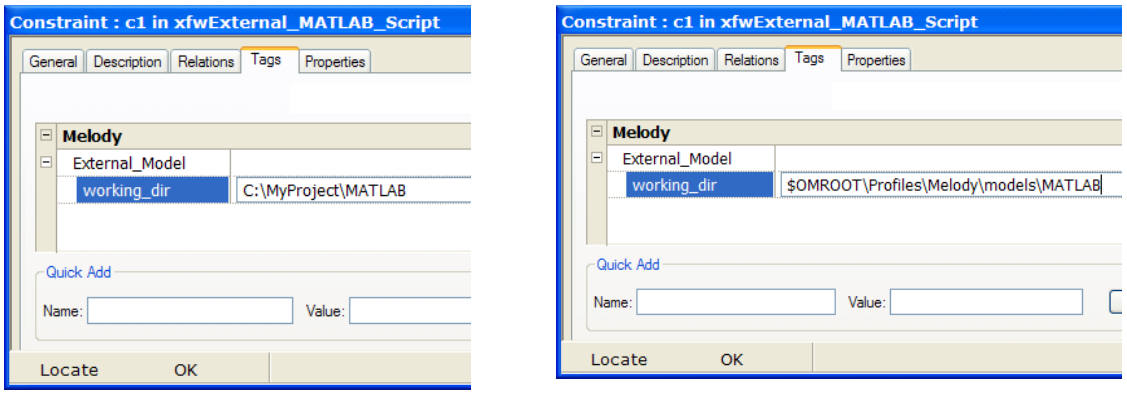


Figure 36: Specifying location of the folder containing the M-file – Absolute (left) and Relative (right)

Step 6. *Type constraint properties by MATLAB constraint blocks and build parametric models*

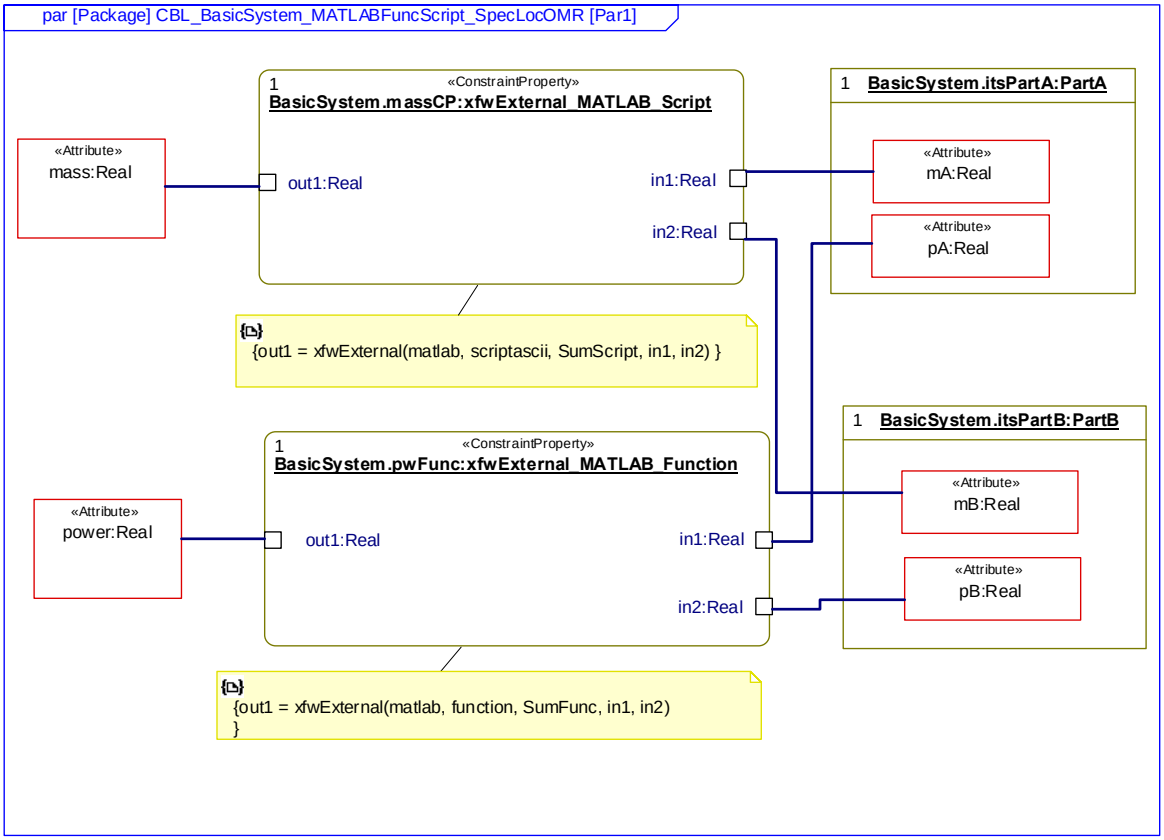


Figure 37: Parametric diagram showing constraint properties massCP & pwFunc of the block BasicSystem typed by constraint blocks xfwExternal_MATLAB_Script & xfw_External_MATLAB_Function respectively.

Figure 37 above illustrates how constraint blocks wrapping MATLAB M-files are used for typing constraint properties of blocks and used in building parametric models.

8.3.1 Using MATLAB scripts

Unlike MATLAB functions, scripts do not have input arguments and output/return values. Melody™ uses intermediate input and output files to transfer SysML instance values (givens) from Rhapsody 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 Melody™, follow the steps below after finishing 0 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 and 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.

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.

How does this work (behind the scene)?
Melody™ writes SysML instance values (from Rhapsody) 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, Melody™ 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 Rhapsody and MATLAB. These are automatically created and managed by Melody™.

8.3.2 Using MATLAB functions

For using Melody™ 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>,...)
```

Follow the step below after completing 0 above.

Step 7. *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 MMC 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.).

The Melody™ - MATLAB/Simulink connection feature is available only with Melody™ Pro edition. For this feature to work, MATLAB must be locally installed on your computer and Mathematica (local or remote) must be selected as the core solver. This feature is not supported if OpenModelica is selected as the core solver.

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

9 TRADE STUDIES

With Melody™ (both Standard and Pro editions), 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 Melody™, trade studies can be now be performed from the earliest stages of system development.

The **LittleEye** tutorial model in your Melody™ installation demonstrates this feature. The model is included here <Rhapsody_Root>\Share\Profiles\Melody\models\tutorials\LittleEye and described in the section 5 of the Melody™ Tutorial document.

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 Melody™.
2. Setup a trade study
   a. Identify trade study inputs, outputs, and constants.
   b. Link inputs and outputs to Excel spreadsheets. Melody™ 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 Melody™ solving purposes—see the Tutorials document for details.

Step 1. Verify that your SysML instance model can be solved in Melody™

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 Melody→Browse. If the Melody™ 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 Melody™ 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, Number of Planes and Number of Crews are the trade study input variables, and Miles Scanner Per 24 Hours is the output variable. Note that values for input/output variables are in columns. Each row,

starting with row 2, represents the different trade study scenarios that will be solved using Melody™.

	A	B	C
1	Number of Planes	Number of Crews	Miles Scanned Per 24 Hours
2	3	4	
3	4	4	
4	5	4	
5	6	4	
6	7	4	
7	3	5	
8	4	5	
9	5	5	

Figure 38: 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 39: 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

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

a) Attributes with causality "given" and Excel access mode "Read" are treated as trade study input variables.
b) Attributes with causality "target" and Excel access mode "Write" are treated as trade study output variables.
c) Attributes 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 attributes 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 Melody→Excel→Setup. This will launch the Excel setup utility, as shown below.

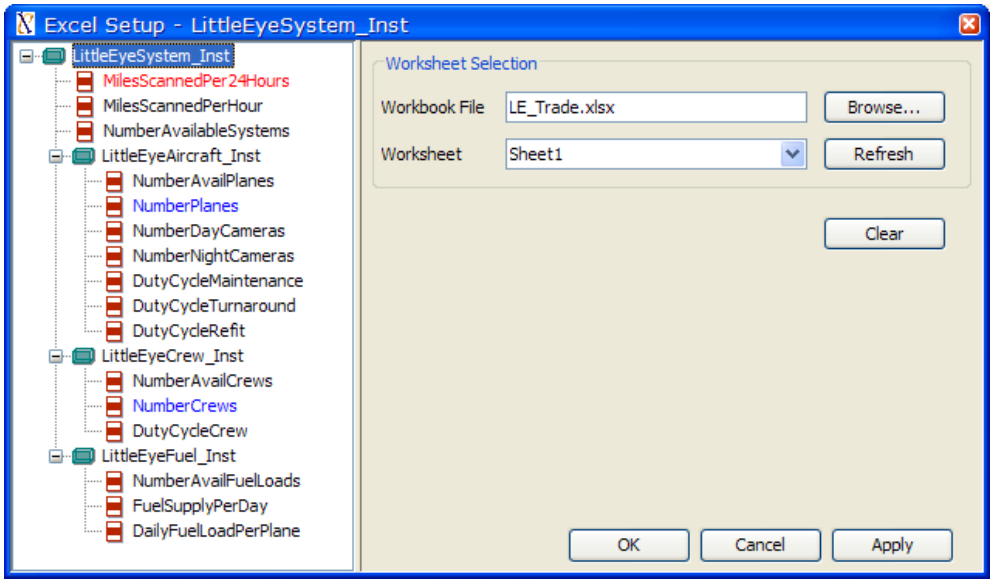


Figure 40: Use the Excel setup utility to link trade study inputs and outputs to spreadsheets

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 attribute 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 41 below. Specify a different workbook/worksheet for the attribute OR use the default from the parent instance (as shown below).

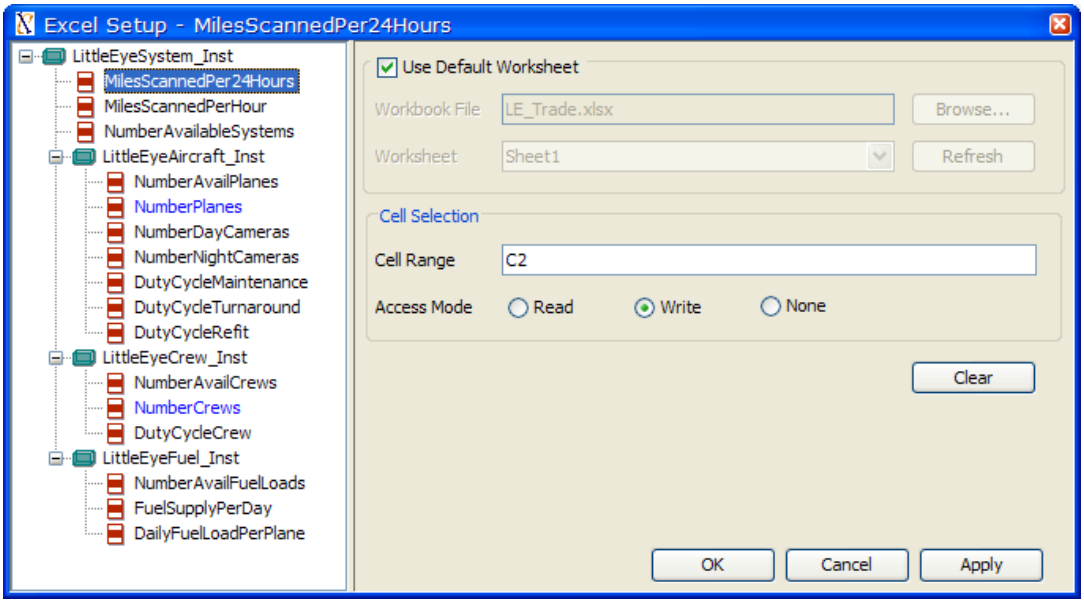


Figure 41: Use the Excel setup utility to link trade study inputs and outputs to spreadsheets

   c) Set the cell range to the cell(s) that contain value(s) for the first scenario. For single-valued attributes, the cell range is a single cell. For multi-valued attributes, the cell range is a set of contiguous cells in a single row. As shown above, the cell range for MilesScannedPer24Hours is set to C2 (spreadsheet shown in Figure 38). Similarly, the cell range for Input Variable 2 and Output Variable 1 (spreadsheet shown in Figure 39) would be E3:G3 and H3:J3 respectively.

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

Step 4. Specify number of scenarios

Right click on the instance package and select Melody→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 Melody™ 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.

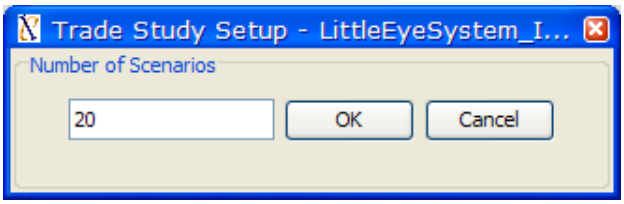


Figure 42: Specify number of scenarios for a trade study

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 Melody→Trade Study→Run. The trade study progress window (as shown in Figure 43) indicates the specific scenario being run and the core solver being used. Completion of a trade study is indicated by the message in Figure 44.

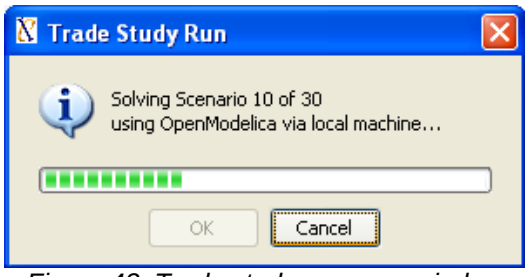


Figure 43: Trade study progress window

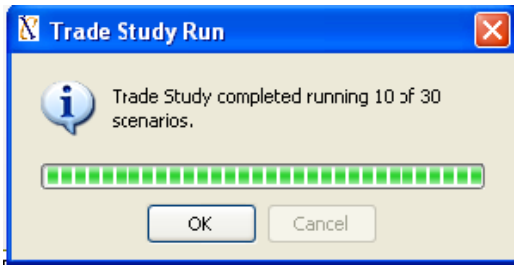


Figure 44: Trade study completion message

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. 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 8.2). SysML parametric models executed in trade studies may include all types of relations as solved using regular Melody™ operation except for custom Mathematica relations that create plots for each scenario[k].

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

9.2 Limitations

The trade study capability in Melody™ R3 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 attributes but not the number of instances in the SysML instance model.

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 Melody™.

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[l] and ModelCenter[m], 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 generated automatically with this version of Melody™. Since trade study outputs are written to Excel spreadsheets, users may leverage the extensive plotting and post-processing capabilities of Excel.

10 COPYRIGHT

10.1 Copyright statement from InterCAX LLC

10.2 Liability disclaimer from InterCAX LLC

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