SysML V2 Cheat Sheet

Version 1.0

This reference sheet gives a concise overview of SysML v2 syntax and concepts for practitioners. **Download and share at**

www.sodiuswillert.biz/sysmlv2

MBSE Best Practices

Start with a methodology – Use a consistent modeling approach (notation + language + method).

Model with purpose – Every element should serve a decision, verification, or communication goal.

Configure your tool early – Define conventions for naming, quantity kinds, etc. and version control before large-scale modeling.

Integrate with your process – Align SysML v2 models with your requirements, test, and change-management workflows.

Keep it small and iterative – Build minimal viable models, validate them with stakeholders, and expand incrementally.

Capture rationale – Record why design choices were made.

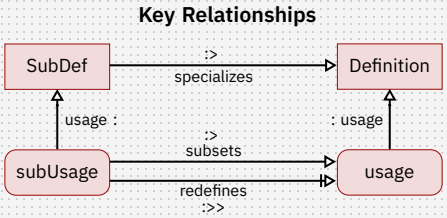
Naming Conventions

- Establish and stick with a naming convention
- Rhapsody can validate naming conventions
- CamelCase is recommended
- Spaces are supported, but discouraged
- Definitions start with UpperCase
- Usages start with lowerCase
- Package usages may be UpperCase
- Alternative <short> names possible

Modeling Guidance

The overview visualizes the three pillars of most MBSE methods: ① requirements, ② behavior and ③ structure. Models have a minimum of two layers, for Ⓐ problem space and Ⓑ solution space, although more layers can be added as needed.

There is no order in which to model. The red arrows are informal to visualize conceptual relationships.



Model Tree

The model tree reflects **ownership**, not just grouping. Moving an element in the tree changes its owner, which can affect semantics.

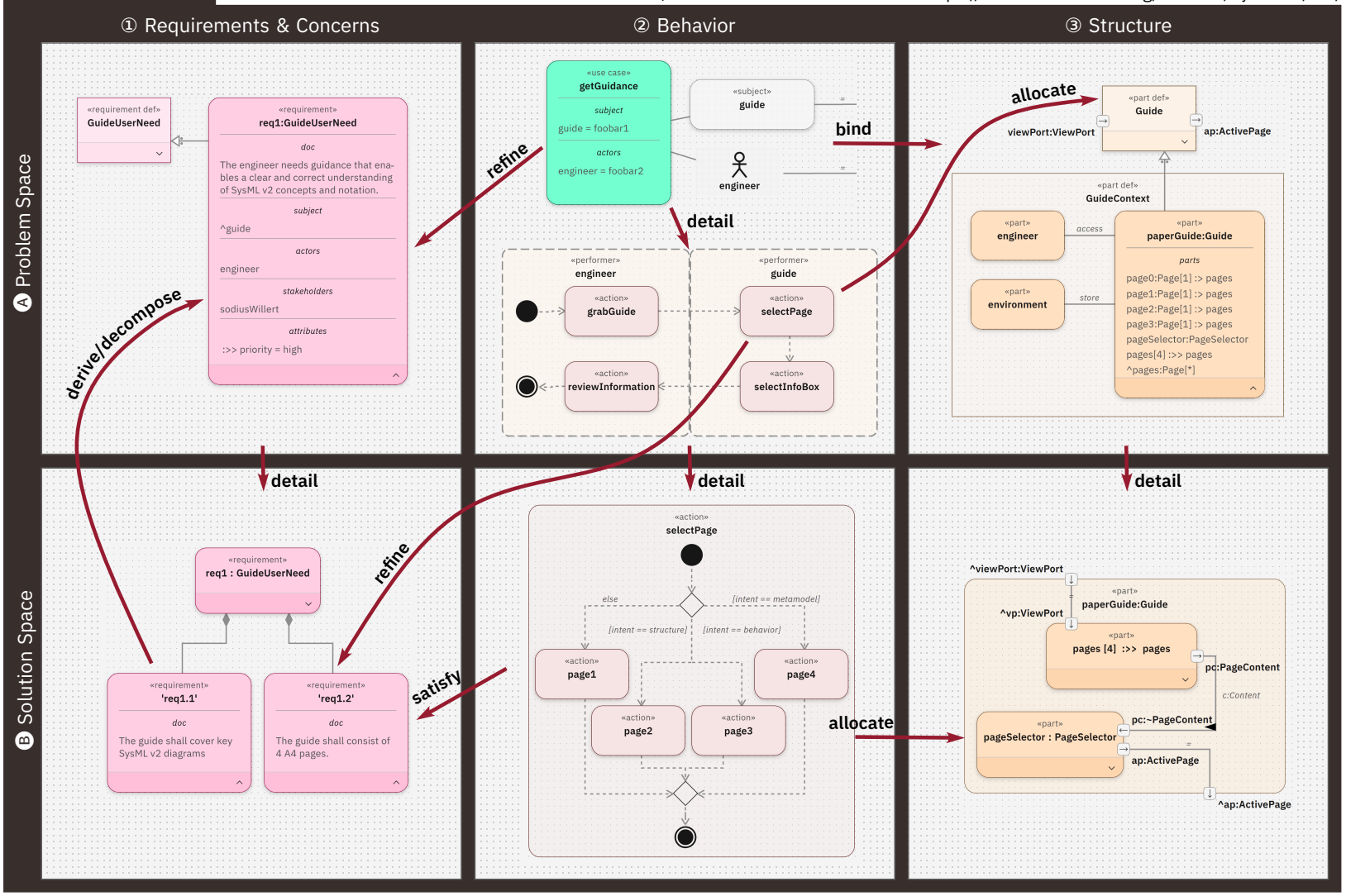
Ensure a **consistent package structure** across all MBSE models, applying the same pattern recursively to subsystems.

Imports define visibility. The tree shows ownership, but imports control what a package can reference.

Choose an **organizing scheme** and follow it rigorously; most methods provide guidance.

Leverage **tool features** such as filtering and query-based containers to keep large models manageable.

Key Relationships



Structure & Domain

Organize your project using packages, namespaces, libraries, and views to ensure scalability and reuse.

Capture your **system's domain** with connected part definitions and parts. Use ports to express the flow of matter, energy, or information across boundaries.

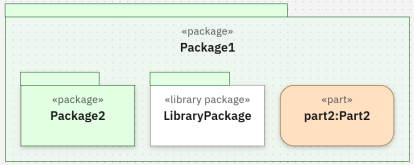
Apply **black-box** and **glass-box** modeling by exposing only ports and interfaces at subsystem boundaries, while keeping internal structure hidden. Define each subsystem with clear external interactions.

Definition & Usage

Definitions declare reusable types. **Usages** reference those types. A definition is explicitly marked with **def**. If def is omitted, it is a usage. Types are normally specified with a **colon** after the usage name, but can also be linked via a **defined by** relationship. Usages without definitions are allowed, but discouraged.

Packages

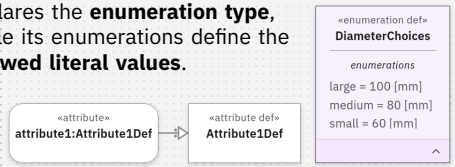
A **namespace** contains elements. A **package** is a kind of namespace for model organization. An **import** relationship allows access of members from other namespaces.



Attributes

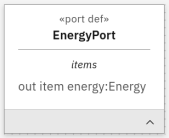
Attributes define **properties of elements**. Any typed element can **own** attributes.

Attributes may use **enumerations** to restrict values to predefined choices. An enumeration def declares the **enumeration type**, while its enumerations define the **allowed literal values**.

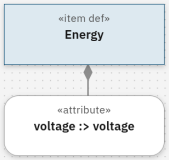


Ports & Interfaces

For connecting elements, the most basic construct is a **connection**, which is also a part. The connection connects two or more occurences which can be nested.



Port definitions define **connection points** to enable interactions and ensure **compatibility**. They specify the allowed kinds of interactions, flows, or signals.



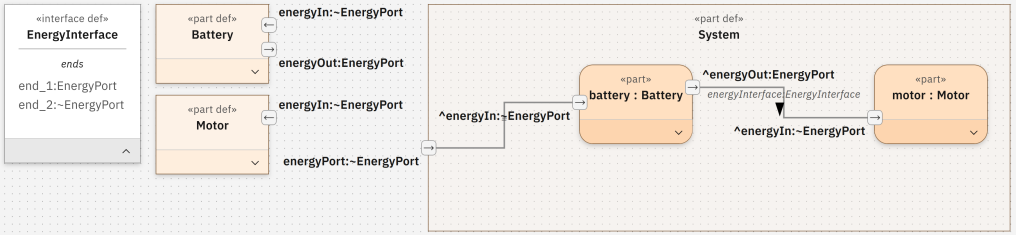
Define **out**, **in** or **inout** attributes for the flows.

By creating appropriate item defs with attributes, you can ensure **compatibility** at the flow level.

An **interface** is a specialized connection that links ports. Ports can be **conjugated (~)**, meaning their direction is reversed. An interface has two or more ends.

The **interface usage** manages the subconnectors that route the flow of the items defined in the connected ports. Tools derive the direction of arrows on the interface automatically.

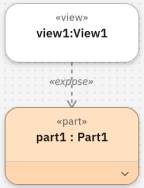
A **binding (=)** enforces equality between two features, typically to synchronize values or parameters across parts. Use a binding (**instead of an interface**) when the outer port is **the same port** as a port on an internal part, e.g. representing a shared physical connector.



Views

A **view** satisfies a **viewpoint** by exposing a portion of the model. There are 8 standard **rendering definitions**:

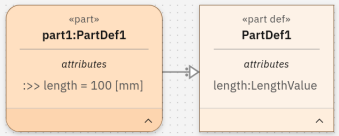
- General (gv)
- Interconnection (iv)
- Action Flow (av)
- State Transition (stv)
- Sequence (sv)
- Geometry (gev)
- Grid (grv)
- Browser (bv)



Quantities & Units

Quantities and units reside in one of several **standard libraries**. The most important ones are **ISQ**, defining physical quantity kinds, and **SI**, which provides the corresponding units.

Pattern: Start with a **quantity attribute** and later assign a **specific value**.



Part & Item Basics

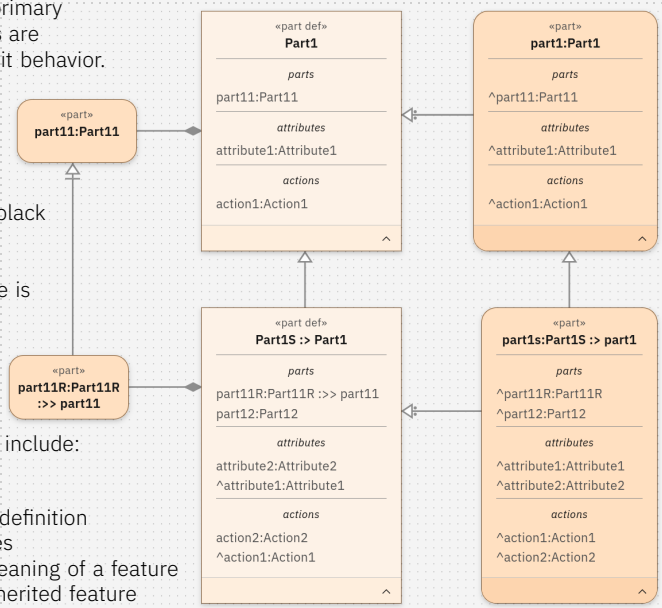
Items (definitions or usages) are the primary structural elements in SysML v2. **Parts** are specializations of items that also exhibit behavior.

Both can contain **child elements**, forming hierarchical system structures.

Containment is represented in diagrams using compartments or a black diamond (composition) indicator.

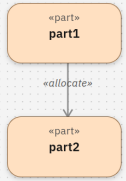
The **^** symbol indicates that the feature is inherited from a parent definition.

Parts support **multiplicity** (1, 0..1, 1..*, etc.). Defaults to any (*).



Common **relationships** involving parts include:

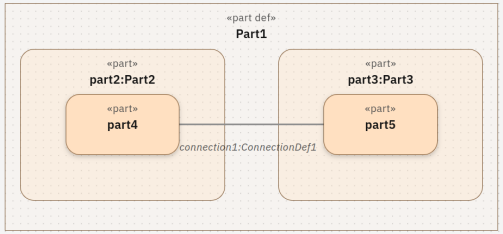
Keyword	Symbol	Description
defined by	:	Links usage to definition
specializes	:>	Inherits features
subsets	:>	Narrows the meaning of a feature
redefines	:>>	Replaces an inherited feature
references	::>	Pointer to a related element



The **allocation** relationship specifies that the target element is responsible for **realizing the intent** of the source.

This relationship plays a key role in **systems engineering**, as it allows "mapping" of elements across the various structures and hierarchies of a system model.

Parts can **contain** other parts (shown as nesting) or be connected with **connectors**. For more advanced connections, use ports and interfaces.



SodiusWillert | IBM | sodiuswillert.biz/sysmlv2

Requirements & Behavior

Analyze your system needs and capture it in **requirements**, **constraints** and **use cases**.

Capture behavior by using **flows** and **actions**. Assign them to performers and visualize them in **swimlanes**.

Support your activities with **validation** and **analysis cases**, or specialized elements like **states** and **time slices**.

Use the right **views**, don't forget **tables**.

SodiusWillert | IBM | sodiuswillert.biz/sysmlv2

Tables

The **table view** is versatile and often used for requirements. Configurability varies between tools.

	Declared name		Nested attribute
1	7.21 Requirement verificationCase1	requirement1.1	
2	requirement1		attribute2 attribute1

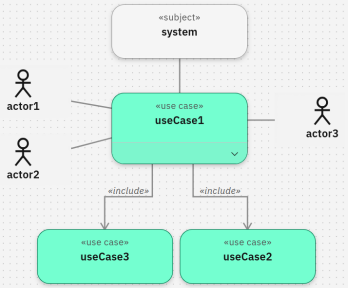
Cases: Use & Others

A **case** is a behaviour model with at least one **subject** and optional **actors**. They have **input** parameters and an **outcome**.

A case can act as the context for specialized cases, like **validation case** or **analysis case**.

Most common is the **use case**, which represents possible scenarios linked to requirements.

Use **definitions** to capture a **reusable case** and create a concrete context with the usage, binding placeholder to concrete elements.



Requirements

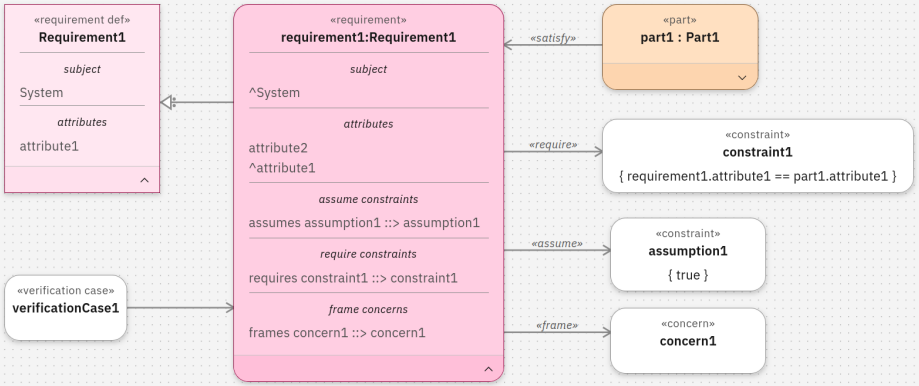
A **requirement** states what the system must achieve. Its **subject** defines the element it applies to, and it may include **attributes** to capture values that can be processed, or other parameters like **stakeholders**.

A requirement may **frame** a concern, indicating the broader stakeholder issue it addresses. Including relevant stakeholders via framing is an **alternative** to adding them to the requirement.

An occurence **satisfies** a requirement when it fulfills what the requirement specifies.

Constraints express conditions that must hold. A requirement can **require** constraints that the system must guarantee, and **assume** constraints that must already be true in the environment.

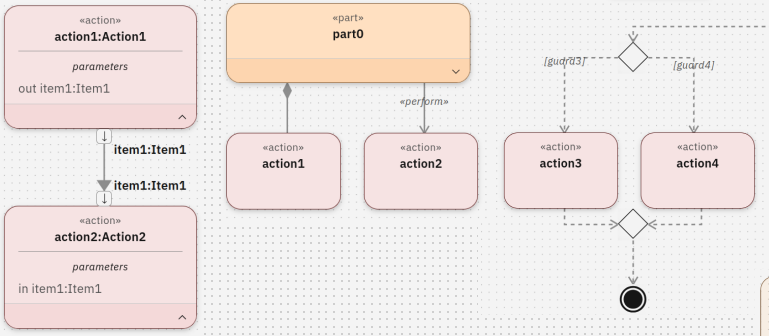
Verification uses a **verification case** to show that a requirement is met.



Flows & Actions

Actions allow the modeling of behavior. A **flow** is an action that connects elements. A **message** is a **flow usage** that can specify the transfer of a payload between source and target.

Actions are performed by **parts**. Parts can own actions directly, or they can reference them via the **perform** relationship.

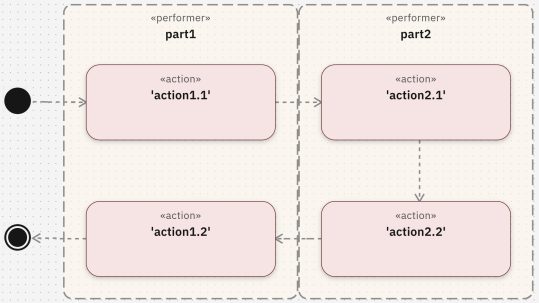


Actions always start with a **start** node and end with a **done** or **terminate** node.

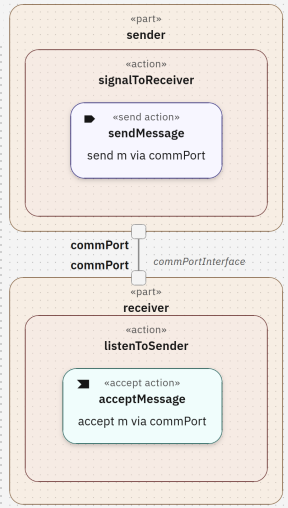
They can run in parallel with **fork** and **join** nodes.

Conditional flows are controlled with **decision** and **merge** nodes that have guards that evaluate as booleans, as well as **loops** of **conditionals**, which require parameters.

The performance can be visualized by placing actions on **swimlanes**.



Communicating with messages can be realized with **send** and **accept** actions. Communication takes place via an **interface** that connects the **ports** from the containing **parts**.

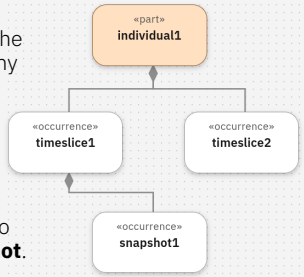


Occurences

An **occurrence** is an element which it is the root element for many SysML v2 elements.

Its lifetime can be partitioned into **time slices**.

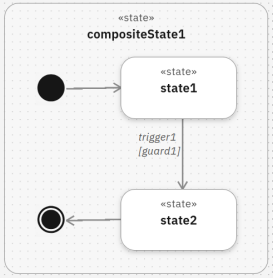
A time slice with zero duration is a **snapshot**.



States

SysML v2 allows the modeling of behavior using **state** machines.

The **trigger** of a transition is an **accept action**. Transitions can require a **guard** condition and may specify an effect action.



Metamodel

KerML defines the foundational meta-model with abstract concepts like types, features, and relationships.

SysML v2 builds on it with systems-engineering constructs such as blocks, ports, and requirements.

The **user model** then applies these to describe a specific system: KerML provides the grammar, SysML v2 the vocabulary, and the user model the description.

KerML Spec

- **KerML** defines the meta-language (precise semantics), SysML v2 defines the domain language (practical vocabulary)
- Everything in KerML is a **typed element**
- Some KerML elements are **used directly** in SysML v2, like relationships
- **Extensibility** depends on KerML

SysML v2 Spec

- Download the **official spec** for free at https://www.omg.org/sysml/sysmlv2/
- It contains many **examples** in textual and graphical notation
- **Separate documents** cover KerML, SyML v1 to v2 transformation, a large automotive sample file and machine readable schemas
- The **API and services** are still in beta (2025)

Elements have several properties, which can be broadly categorised like this:

- **Essential** aspects like name or description
- **Other** specific aspects like abstract, variation
- **Owned** relationships (child elements)
- **References** to and from other elements
- **Metadata** associated with the element
- **Links** to the development ecosystem via URIs

Textual Notation

- **The model**, not the text, remains the single source of truth
- Textual notation is just **another view** of the same model
- It **omits** layout, derived relationships, and internal element IDs
- It's human **readable**, **diffable**, and suitable for **version control**
- **Round-tripping** between text and model is usually possible
- **Tools vary**: model-first, text-first, or fully equivalent approaches
- Many tools **mix both**, e.g., textual compartments in diagrams

- **Enables** automation, scripting, and CI/CD integration for models

API

- The SysML v2 API provides structured, **tool-independent access** to the semantic model
- It **enables** automation, transformation, and integration with **external tools**
- Most tools offer **endpoints** or **SDKs**
- The **API** and services spec is **still in beta** (as of December 2025)

Model Element Inheritance

This diagram visualizes the **simplified** inheritance structure of the primary model elements. It is intended as guidance and does not use a formal notation on purpose, to prevent being misleading.

Due to the nature of SysML v2, this visualization does not differentiate **definition** and **usage**. We ommitted **function** and **association** hierarchy. **Root elements** are still connected via the KerML hierarchy, which is not shown for simplicity.

The numbers reference the chapter in the SysML v2 specification.

The **yellow elements** are frequently used and are sufficient to start modeling.