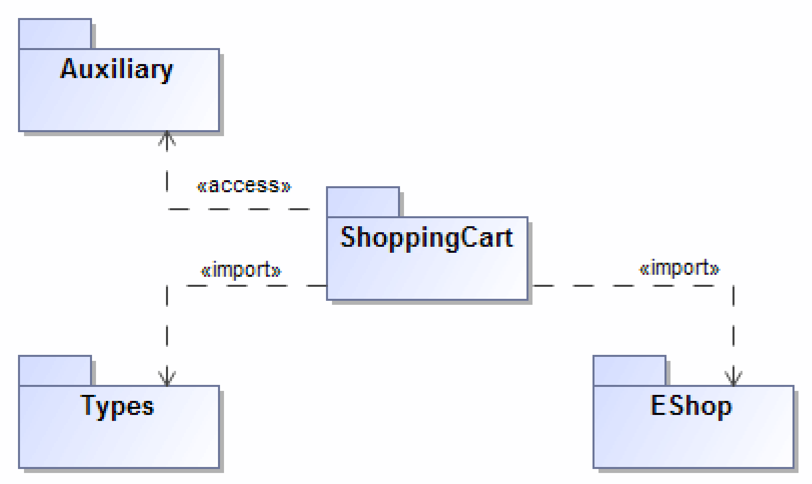
Dependency

A dependency is a relationship signifying that a model element requires other model elements for its specification or implementation. The client element is either semantically or structurally dependent on the definition of the supplier element(s).

A dependency is shown as a dashed arrow between classes or packages. The client element (at the tail of the arrow) depends on the supplier element (at the arrowhead). You can label the arrow with an optional stereotype and an individual name.

You can also draw a dependency between a Class and other Class elements, such as attributes and operations.



Example of dependency relationships: «access» and «import».

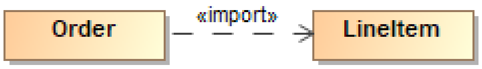
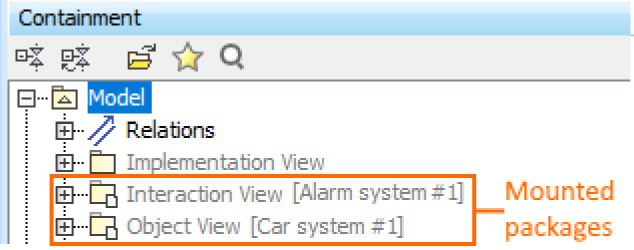
Dependency, abstraction, and usage relationships are defined in the dialog of the same structure. They differ from one another only by the corresponding Specification name. You can specify a dependency by changing its property values in the dependency Specification window. Each property is described in the description area on this window.

Editing property values

For more information about specifying property values, see Editing property values.

There are several kinds of dependencies:

Dependency kind	Description
Template binding	Represents a relationship between a templatable element and a template. A template binding specifies the substitutions of actual parameters for the formal parameters of the template. You can specify template binding dependency properties and find each property's description in the template binding dependency Specification window. Descriptions are presented in the description area of the Specification window.
Abstraction	Relates two elements or sets of elements that represent the same concept at different levels of abstraction or from different viewpoints. In the metamodel, an abstraction is a dependency in which there is a mapping between the supplier and the client. Define an abstraction relationship in the Abstraction Specification window.
Usage	A relationship in which one element requires another element (or set of elements) for its full implementation or operation. In the metamodel, a usage is a dependency in which the client requires the presence of the supplier. Define a usage relationship in the Usage Specification window.
Package merge	A directed relationship between two packages, indicating that the contents of the two packages are to be combined. It has a dependency relationship with the applied stereotype *«merge»*. Define a merge relationship in the Dependency Specification window.

Element import	A directed relationship between an importing namespace and a packageable element. The name of the packageable element or its alias is added to the namespace of the importing namespace. It has a dependency relationship with the applied stereotype *«import»*. Define an import relationship in the Dependency Specification window. To draw the Element Import link, select the Element Import path to draw in the Class diagram toolbar from the Abstraction group. 
Access	Shows that elements can only be accessed from a package, and it cannot be referenced.
Deployment	The allocation of a deployment target to an artifact or artifact instance.
Role Binding	A relationship in a Composite Structure diagram. Role Binding is used to connect Parts with Collaboration Use.
Mount	A mount dependency represents a former location of a shared package; that is, a location where the package was mounted in a local project or a Teamwork Server project. A mount dependency is created: • After you export a project from Teamwork Server to Teamwork Cloud; or • When a local project is added to Teamwork Cloud. ✓ Mounted packages in the Containment tree are adorned with an icon and have their project usage name in brackets, as shown in the following figure: The purpose of establishing the mount dependency is to ensure the same containment structure during project transformation to Teamw validation rules, tables etc.  Containment Model Relations Implementation View Interaction View [Alarm system #1] Object View [Car system #1] Mounted packages

Related pages

- Working with depe
- Package import
- Abstraction