

CHAPTER 15

Extension Mechanisms

Extension mechanisms are the means for customizing and extending the UML. The model elements described in Chapters 6 through 14 are instances of metamodel elements that can be considered the core of UML diagrams. (See Chapter 5 for a description of the metamodel.) The UML defines properties for each model element and a system of stereotypes and constraints that permit you to add new elements. This chapter describes the basic extension mechanisms and two particular extensions that have already proven useful: one for software engineering and one for business modeling.

Stereotypes

Stereotypes (Figure 15-1) are a mechanism for classifying or marking model elements and introducing new types of modeling elements.

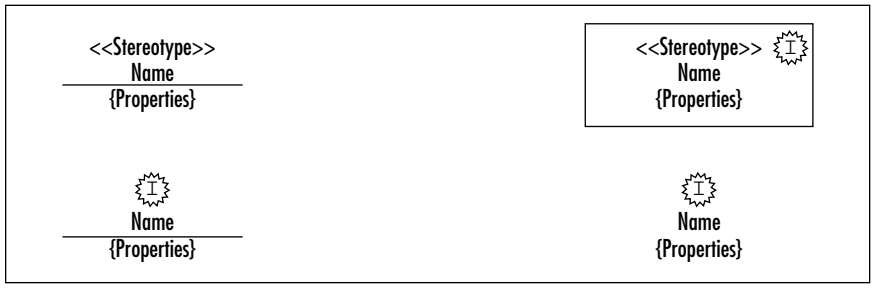


Figure 15-1: Stereotypes

Stereotypes

- Are strings enclosed in guillemets (<< >>), or double angle brackets, preceding the name of an element. A property may follow the name of the element.

- Are limited to one per model element.
- May be elements within a list of strings. The stereotype applies to all succeeding elements of the list until another stereotype keyword list element appears, an empty stereotype keyword nullifies it, or the end of the list is reached.
- May be used for classifying or marking model elements, in which case stereotypes are depicted when the element is rendered.
- May be used for introducing new types of modeling elements. New modeling elements
  - Must be subclasses of existing modeling elements.
  - Must have the same form or characteristics (attributes and operations) as their base metamodel classes. That is, they inherit the characteristics of their base classes via a generalization relationship.
  - Must have a different intent or semantics than their base classes. That is, they have a different meaning or purpose then their base classes.
  - May extend their base classes by adding properties (constraints and tagged values) to the characteristics they receive from their base classes. These properties are implicitly attached to model elements of the new stereotype. If tagged values are left unspecified in the new modeling element, they must be specified when the stereotype is attached to a model element.
  - May have an associated graphic icon or graphic marker.
  - May be rendered using the base model element symbol in addition to the stereotype keyword with or without the graphic icon, or rendered using the graphic icon and collapsing the base model element symbol.
  - May be described in a classification hierarchy that extends the metamodel hierarchy. This is a user-defined metamodel diagram that describes how stereotypes extend the UML metamodel. These classes must themselves be stereotyped using the "stereotype" keyword. The attributes of these classes may be referenced by model elements using the attribute names as keywords in tagged values. Properties and attributes must not conflict with those that a stereotype inherits.

Standard Stereotypes

Table 15-1 shows the standard stereotype elements predefined in the UML.

Table 15-1: Standard Stereotypes

Name	*Application*	*Semantics*
<<actor>>	Class	The class defines a set of external objects that interact with a system.
<<association>>	Association role end	The corresponding element is accessible via an association.

Table 15-1: Standard Stereotypes (continued)

Name	*Application*	*Semantics*
<<becomes>>	Dependency	The dependency is between a source instance and a target instance. The dependency specifies that the source and target represent the same instance at different points in time, but with different state and roles. That is, the source instance becomes the target instance with possibly different state and role.
<<bind>>	Dependency	The dependency is between a source class and a target template. The dependency specifies that the class is created by binding actual values to the formal parameters of the template.
<<call>>	Dependency	The dependency is between a source operation and a target operation. The dependency specifies that the source operation invokes the target operation. The target operation must be accessible or within the scope of the source operation.
<<constraint>>	Note	The note is a constraint.
<<constructor>>	Operation	The operation creates an instance of the classifier to which the operation is attached. This is equivalent to the "create" stereotype. See the "create" and "destructor" stereotypes.
<<classify>>	Dependency	The dependency is between a source instance and a target classifier. The dependency specifies that the source instance is an instance of the target classifier. See the "declassify" stereotype.
<<copy>>	Dependency	The dependency is between a source instance and a target instance. The dependency specifies that the source and target represent different instances with the same state and roles. The target instance is an exact copy of the source instance. The source and target are unrelated after the copy.

Table 15-1: Standard Stereotypes (continued)

Name	*Application*	*Semantics*
<<create>>	Operation	The operation creates an instance of the classifier to which the operation is attached. The operation is a constructor. This is equivalent to the "constructor" stereotype. See the "constructor" and "destroy" stereotypes.
	Event	The event denotes that an instance of the class enclosing the state machine is created. The event may be applied only to an initial transition at the topmost level of the state machine.
<<declassify>>	Dependency	The dependency is between a source instance and a target classifier. The dependency specifies that the source instance is no longer an instance of the target classifier. See the "classify" stereotype.
<<destroy>>	Operation	The operation destroys an instance of the classifier to which the operation is attached. This is equivalent to the "destructor" stereotype. See the "destructor" and "create" stereotypes.
	Event	The event denotes that an instance of the class enclosing the state machine is destroyed.
<<delete>>	Refinement	The refinement is between a source element and a target element. The refinement specifies that the source element can no longer be refined.
<<derived>>	Dependency	The dependency is between a source element and a target element. The dependency specifies that the source element is derived from the target element. That is, the source element is not an instance of the target element, but an instance of another element that is a subclass or subtype of the target element.
<<destructor>>	Operation	The operation destroys an instance of the classifier to which the operation is attached. This is equivalent to the "destroy" stereotype. See the "destroy" and "constructor" stereotypes.
<<document>>	Component	The component represents a document.

Table 15-1: Standard Stereotypes (continued)

Name	*Application*	*Semantics*
<<enumeration>>	Data type	The data type specifies a set of identifiers that are the possible values of an instance of the data type.
<<executable>>	Component	The component represents an executable program that may be run on a node.
<<extends>>	Generalization	The generalization is between a source use case and a target use case. The generalization specifies that the contents of the source use case may be added to the target use case. The relationship specifies where the contents should be added (extension point) and the condition that must be satisfied in order to add the source use case. When an instance of the target use case reaches the extension point and the condition is fulfilled, the instance continues according to a sequence that is the result of extending the target use case sequence with the source use case sequence at that point.
<<facade>>	Package	The package only contains references to model elements owned by other packages. It does not contain any model elements of its own. It is used to provide a "public view" of some of the contents of another package.
<<file>>	Component	The component represents a document or file containing source code or data.
<<framework>>	Package	The package consists mainly of patterns (collaborations).
<<friend>>	Dependency	The dependency is between a source element (operation, class, or package) and a target element (operation, class, or package) in a different package. The dependency specifies that the source element may access the target element regardless of the declared visibility of the target element.
<<global>>	Association role end	The corresponding element is accessible because it has global scope within the context in which it is accessed. That is, the instance is accessible throughout the system.

Table 15-1: Standard Stereotypes (continued)

Name	*Application*	*Semantics*
<<import>>	Dependency	The dependency is between a source package and a target package. The dependency specifies that the source package receives and may access the public contents of the target package.
<<implementation class>>	Class	The class defines the implementation of another class in some programming language. The class is not a type. An instance may have zero or one implementation classes. But an instance may statically have many general or non-implementation classes, and it may dynamically gain or lose general classes over time.
<<inherits>>	Generalization	The generalization is between a source classifier and a target classifier. This constrains the generalization such that instances of the source classifier may not be substituted for instances of the target classifier. See the "subclass" and "subtype" stereotypes.
<<instance>>	Dependency	The dependency is between a source instance and a target classifier. The dependency specifies that the source instance is an instance of the target classifier.
<<interface>>	Class	The class defines a collection of operations that may be used for defining a service offered by other classes. The class may contain only externally accessible, or public, operations without methods.
<<invariant>>	Constraint	The constraint is attached to a set of classifiers or relationships. The constraint specifies a condition that must hold true for the classifiers or relationships and their instances. See the "precondition" and "postcondition" stereotypes.
<<local>>	Association role end	The corresponding element is accessible because it has local scope within the context in which it is accessed. That is, the instance is a local variable within an operation.

Table 15-1: Standard Stereotypes (continued)

Name	*Application*	*Semantics*
<<library>>	Component	The component represents a static or dynamic library. A static library is utilized by a program during development time: the library is linked into the program. A dynamic library is utilized by a program during run time: the library is accessed by the program during execution.
<<metaclass>>	Classifier	The class is a metaclass of some other class.
	Dependency	The dependency is between a source classifier and a target classifier. The dependency specifies that the target classifier is the metaclass of the source classifier.
<<parameter>>	Association role end	The corresponding element is accessible because it is a parameter within the context in which it is accessed. That is, the instance is a parameter variable within an operation.
<<postcondition>>	Constraint	The constraint is attached to an operation. The constraint specifies a condition that must hold true after the invocation of the operation. See the “invariant” and “precondition” stereotypes.
<<powertype>>	Classifier	The classifier is a metatype whose instances are subtypes of another type. That is, the classifier is the type of a discriminator involved in a generalization relationship.
	Dependency	The dependency is between a source set of generalizations and a target classifier. The dependency specifies that the target classifier is the powertype of the source set of generalizations. That is, the classifier is the type of the discriminator involved in the set of generalizations.
<<precondition>>	Constraint	The constraint is attached to an operation. The constraint specifies a condition that must hold true for the invocation of the operation. See the “invariant” and “postcondition” stereotypes.

Table 15-1: Standard Stereotypes (continued)

Name	*Application*	*Semantics*
<<private>>	Generalization	The generalization is between a source classifier and a target classifier. The generalization specifies that inherited features of the target classifier are hidden, or made private, within the source classifier. Instances of the source classifier may not be substituted for instances of the target classifier.
<<process>>	Classifier	The classifier denotes an active class that has a heavyweight flow of control. It is a thread with a representation of control (code), and may be composed of threads. See the "thread" stereotype.
<<query>>	Operation	The operation does not modify the state of the instance. See the "update" stereotype.
<<realize>>	Generalization	The generalization is between a source element and a target element. The generalization specifies that the source element realizes the target element. If the target element is an implementation class, the relationship implies inheritance of operations but not of structure (attributes and associations). If the target element is an interface, the source element supports the operations of the interface.
<<refine>>	Dependency	The dependency is between a source element and a target element. The dependency specifies that the two elements are at different semantic levels of abstraction. The source element refines or is derived from the target element. This may be used between different types of models (analysis, design, and implementation) or between use cases and their implementations. The refinement may be described in a note attached to the dependency.
<<requirement>>	Comment	The note specifies a responsibility or obligation of the element to which it is attached.
<<self>>	Association role end	The corresponding instance is accessible because it is the requester.
<<send>>	Dependency	The dependency is between a source operation and a target signal class. The dependency specifies that the operation sends the signal.

Table 15-1: Standard Stereotypes (continued)

Name	*Application*	*Semantics*
<<signal>>	Class	The class defines a signal whose name may be used to trigger transitions. The parameters of the signal are shown in the attribute compartment. The class must not have any operations. It may have generalization relationships to other signal classes.
	Operation	The class accepts and responds to a given signal. The response (operation) is shown using a state machine.
<<stereotype>>	Classifier	The classifier is a stereotype. The classifier is a metamodel class. This is used to model stereotype hierarchies.
<<stub>>	Package	The package is not completely transferred to other packages via generalization relationships. That is, only the public parts of the package are inherited, and the protected parts of the package are not inherited.
<<subclass>>	Generalization	The generalization is between a source classifier and a target classifier. This constrains the generalization such that instances of the source classifier may not be substituted for instances of the target classifier. See the "inherits" and "subtype" stereotypes.
<<subtraction>>	Refinement	The refinement specifies that the target can no longer be refined.
<<subtype>>	Generalization	The generalization is between a source classifier and a target classifier. This indicates that instances of the source classifier may be substituted for instances of the target classifier. This is the default semantic for generalizations. See the "inherits" and "subclass" stereotypes.
<<subsystem>>	Package	The package is a subsystem. It has one or more public interfaces and a private implementation that realizes the interfaces. It must have at least one public interface and must not make any of its implementation publicly accessible.
<<supports>>	Dependency	The dependency is between a source node and a target component. The dependency specifies that the component may reside on the node; that is, the node supports or allows the component to execute on the node.

Table 15-1: Standard Stereotypes (continued)

Name	*Application*	*Semantics*
<<system>>	Package	This package denotes a collection of models that describe a system from different viewpoints. Each model shows a different view of the system, including specification and realization views. This package is the root of the package hierarchy. It may be contained only in a system package.
<<table>>	Component	The component represents a database table.
<<thread>>	Classifier	The classifier denotes an active class that has a lightweight flow of control. It is a single path of execution through some representation of control (code). See the "process" stereotype.
<<top level package>>	Package	The package denotes the topmost package in a model. It represents all the nonenvironmental parts of the model. It is at the top of the containment hierarchy in a model.
<<trace>>	Dependency	The dependency is between a source element and a target element. The dependency specifies that the two elements represent the same concept at different levels of semantics or from different points of view. This may be used for tracing requirements or to track changes of model elements.
<<type>>	Class	The class specifies a set of instances together with the operations applicable to the objects. The class may have attributes, operations, and associations, but it may not contain any methods.
<<update>>	Operation	The operation modifies the state of the instance. See the "query" stereotype.
<<use case model>>	Package	The package denotes a model that describes a system's functional requirements. It contains use cases and their interactions with actors. The package may contain only use cases, actors, extends and uses relationships between use cases, communication relationships between use cases and actors, and generalization relationships between actors.

Table 15-1: Standard Stereotypes (continued)

Name	*Application*	*Semantics*
<<uses>>	Generalization	The generalization is between a source use case and a target use case. The generalization specifies that the contents of the target use case are included or used in the description of the source use case. The relationship is used for extracting shared behavior.
	Dependency	The dependency is between a source element and a target element. The dependency specifies that the source element requires the presence of the target element for its correct implementation or functioning.
<<utility>>	Classifier	The classifier denotes a named collection of nonmember attributes and operations. All the instance scope attributes and operations have class scope by default.

Properties

Properties (Figure 15-2) are characteristics of an element.



Figure 15-2: Property

Properties

- Are strings consisting of a comma-separated list of substrings (property strings) enclosed in curly braces. Each substring is a value attached to the element to which the whole property string is attached. These values may include attributes, associations, constraints, and tagged values.
- May be elements within a list of strings. The properties apply to all succeeding elements of the list until another property string list element appears or the end of the list is reached. Properties (applied to a list of elements) are not superseded by property strings attached to individual elements, but may be augmented or modified by them.
- May be associated with a single graphical symbol, where the string
  - Is a property of the single graphical symbol to which it is attached.
  - May follow the name of an element (Figure 15-1).
  - May be placed in a note and attached (using a dashed line) to the graphical symbol (Figure 15-3). If the note is stereotyped, the property need not be enclosed in curly brackets.

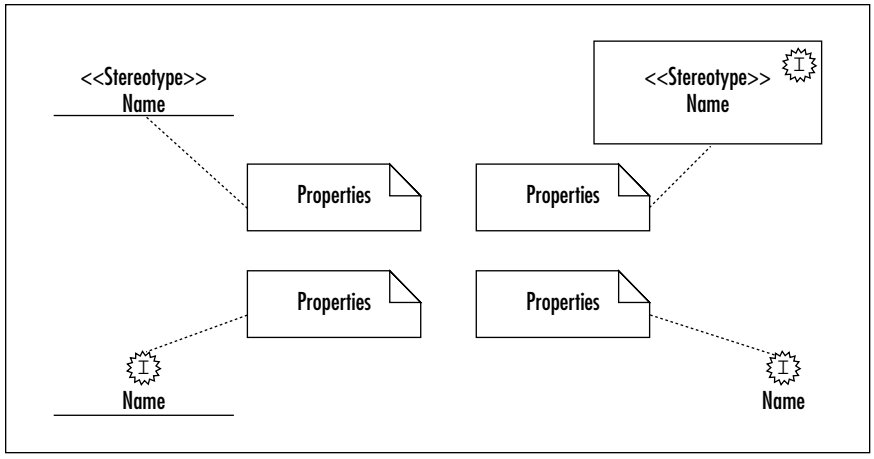


Figure 15-3: Properties for Symbols Using Notes

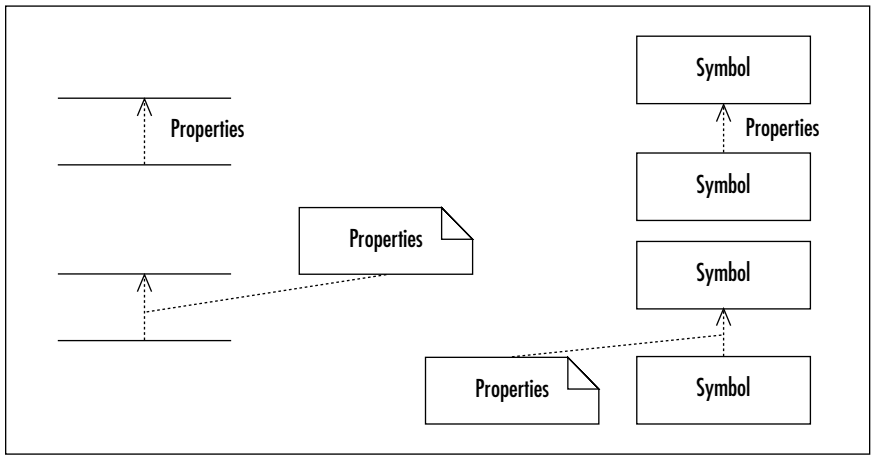


Figure 15-4: Properties between Two Symbols

- May be associated with two graphical symbols (Figure 15-4), where the string
  - Is a property of the two graphical symbols to which it is attached.
  - May be attached to a dashed arrow from one element to the other element. The direction of the arrow is relevant within the property. If the property is a constraint, the direction of the arrow indicates that one element is dependent on another element.
  - May be placed in a note and attached (using a dashed line) to a dashed arrow from one element to the other element. The direction of the arrow is relevant. If the note is stereotyped, the property need not be enclosed in curly brackets.

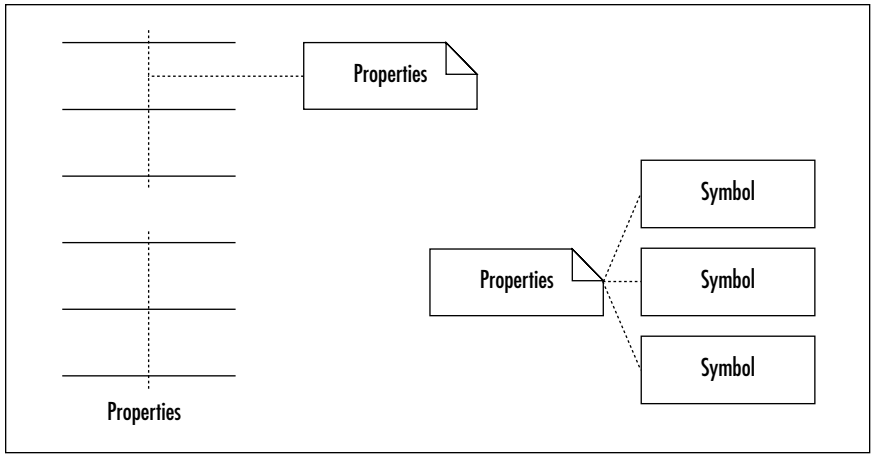


Figure 15-5: Properties between Three or More Symbols

- May be used with three or more graphical symbols (Figure 15-5), where the string
  - Is a property of the three or more graphical symbols to which it is attached.
  - May be attached to a dashed line crossing a set of paths. This indicates that the property applies to the paths the dashed line crosses.
  - May be placed in a note and attached (using a dashed line) to a dashed line crossing a set of paths. If the note is stereotyped, the property need not be enclosed in curly brackets.
  - May be placed in a note and attached (using a dashed line) to three or more graphical symbols. If the note is stereotyped, the property need not be enclosed in curly brackets.

Constraints

Constraints (Figure 15-6) are a mechanism for specifying semantics or conditions that must be maintained as true for model elements.



Figure 15-6: Constraint

Constraints

- Are strings within property strings. Constraint strings specify conditions that must be maintained as true for a model element or elements.

- Must be expressed using a particular language (computer language, human language, or the Object Constraint Language) that is conceptually interpretable or executable.
- May be comments. Comments are text strings written in a human language. The meaning of the string is significant to humans.

Standard Constraints

Table 15-2 shows the standard constraint elements predefined in the UML.

Table 15-2: Standard Constraints

Name	*Application*	*Semantics*
abstract	Class	The class has at least one abstract operation and may not be instantiated.
	Operation	The operation provides an interface specification, but no implementation.
active	Object	The object owns a thread of control and may initiate control activity.
add only	Association end	Additional links may be added (the multiplicity is variable), but links may not be modified or deleted. See the “frozen” constraint.
association	Association end	The corresponding instance is accessible via an association.
broadcast	Operation signal	The request is simultaneously sent to multiple instances in an undefined and unspecified order.
class	Attribute	The attribute has class scope. All instances of a class share one value for the attribute. See the “instance” constraint.
	Operation	The operation has class scope. The operation may be applied on the class. See the “instance” constraint.
complete	Generalization	For a set of generalizations, all subtypes have been specified, some may be suppressed or elided, and no additional subtypes are permitted. See the “incomplete” constraint.
concurrent	Operation	The operation may be called from concurrent threads simultaneously, and all threads may proceed concurrently. Concurrent operations perform correctly in the case of simultaneous sequential or guarded access. See the “sequential” and “guarded” constraints.

Table 15-2: Standard Constraints (continued)

Name	*Application*	*Semantics*
destroyed	Class role Association role	The model element is destroyed during the execution of an interaction. See the "new," "new destroyed," and "transient" constraints.
disjoint	Generalization	For a set of generalizations, instances may have no more than one of the given subtypes as a type. Derived classes may not have generalization relationships to more than one of the subtypes. This is the default semantic for generalizations. See the "overlapping" constraint.
frozen	Association end	No links may be added, deleted, or moved from an object after the object is created and initialized. See the "add only" constraint.
guarded	Operation	The operation may be called from concurrent threads simultaneously, but only one thread is allowed to commence. Other calls are blocked until the performance of the first call is complete. Guarded operations perform correctly in the case of simultaneous sequential access. See the "sequential" and "concurrent" constraints.
global	Association end	The corresponding instance is accessible because it has global scope within the context in which it is accessed. That is, the instance is accessible throughout the system.
implicit	Association	The association is only notational or conceptual and is not utilized in elaborating the model (manifest).
incomplete	Generalization	For a set of generalizations, not all subtypes have been specified, some may be suppressed or elided, and additional subtypes are permitted. This is the default semantic for generalizations. See the "complete" constraint.
instance	Attribute	The attribute has instance scope. Each instance of a class has its own value for the attribute. See the "class" constraint.

Table 15-2: Standard Constraints (continued)

Name	*Application*	*Semantics*
	Operation	The operation has instance scope. The operation may be applied on instances of a class. See the "class" constraint.
local	Association end	The corresponding instance is accessible because it has local scope within the context in which it is accessed. That is, the instance is a local variable within an operation.
new	Class role Association role	The model element is created during the execution of an interaction. See the "destroyed," "new destroyed," and "transient" constraints.
new destroyed	Class role Association role	The model element is created and destroyed during the execution of an interaction. This is equivalent to the "transient" stereotype. See the "new," "destroyed," and "transient" constraints.
or	Association	From a set of associations, only one is manifest for each associated instance.
ordered	Association end	The corresponding elements form an ordered set in which duplicates are prohibited. See the "unordered" constraint.
overlapping	Generalization	For a set of generalizations, instances may have more than one of the given subtypes as a type. Derived classes may have generalization relationships to more than one of the subtypes. See the "disjoint" constraint.
parameter	Association end	The corresponding instance is accessible because it is a parameter within the context in which it is accessed. That is, the instance is a parameter variable within an operation.
polymorphic	Operation	The operation may be overridden by subclasses.
private	Attribute Operation	The feature (attribute or operation) is not accessible outside its class, and is not accessible by descendants of its class. See the "public" and "protected" constraints.

Table 15-2: Standard Constraints (continued)

Name	*Application*	*Semantics*
protected	Attribute Operation	The feature (attribute or operation) is not accessible outside its class, but is accessible by descendants of its class. See the "public" and "private" constraints.
public	Attribute Operation	The feature (attribute or operation) is accessible outside its class, and is accessible by descendants of its class. See the "protected" and "private" constraints.
query	Operation	The operation does not modify the state of the instance. See the "update" constraint.
self	Association end	The corresponding instance is accessible because it is the requester.
sequential	Operation	The operation may be called from concurrent threads simultaneously, but the callers of the operation must coordinate so that only one call to the operation may be outstanding at any one time. See the "guarded" and "concurrent" constraints.
sorted	Association end	The corresponding elements are sorted based on their internal values. This specifies a design decision for implementation.
transient	Class role Association role	The model element is created and destroyed during the execution of an interaction. This is equivalent to the "new destroyed" stereotype. See the "new," "destroyed," and "new destroyed" constraints.
unordered	Association end	The corresponding elements form an unordered set in which duplicates are prohibited. This is the default constraint for association ends. See the "ordered" constraint.
update	Operation	The operation modifies the state of the instance. See the "query" constraint.
vote	Operation Signal	The return value of the request is selected by a majority vote of all the values returned from multiple instances.

Tagged Values

Tagged values (Figure 15-7) are a mechanism for specifying properties of model elements.

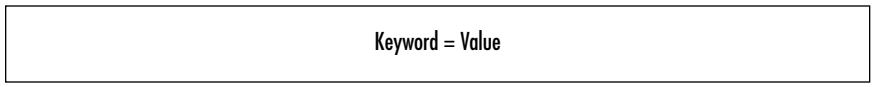


Figure 15-7: Tagged Value

Tagged values

- Are keyword-value pairs within property strings. These keyword-value pairs are known as *property specifications*. Keyword-value pairs specify values for model element properties.
- Must have a keyword. Keywords
  - Are called *tags*.
  - Are strings that represent properties of the element to which the keyword-value pair is attached.
  - Must appear only once for a given element.
  - May represent tagged values for a model element. These are attributes defined for the model element.
  - May represent attributes defined by modeling elements. These are attributes defined by the metaclass of the model element.
  - Must be specified if no default value is specified by an element's meta-model class.
- May have an equal sign followed by a value following the keyword. Values
  - Are strings that represent the value of the keyword or property.
  - May be omitted (along with the equal sign) if the property is of type Boolean. When the keyword is present, the property is true. When the keyword is omitted, the property is false.

Standard Tagged Values

Table 15-3 shows the standard tagged value elements predefined in the UML.

Table 15-3: Standard Tagged Values

Name	*Application*	*Semantics*
documentation	Element (any modeling element)	The tag value specifies a comment, description, or explanation of the element.
location	Classifier	The tag value specifies the component in which the classifier is located.
	Component	The tag value specifies the node on which the component is located.

Table 15-3: Standard Tagged Values (continued)

Name	*Application*	*Semantics*
persistence	Attribute Classifier Instance	The tag specifies that the model element is persistent. If an element is transitory, its state is destroyed when it or its container is destroyed. If an attribute is persistent, its state is not destroyed when it or its container is destroyed, and may be recalled at a later time.
responsibility	Classifier	The tag value specifies a contract by or an obligation of the classifier.
semantics	Classifier	The tag value specifies the meaning of the classifier. That is, it specifies what the classifier represents and its purpose.
	Operation	The tag value specifies the meaning of the operation. That is, it specifies what the operation represents and its purpose.

UML Extension for the Objectory Process for Software Engineering

The UML Extension for the Objectory Process for Software Engineering document registers a process-specific extension to the UML. It includes a set of stereotypes (Table 15-4), various association rules (Table 15-5), and a set of stereotype icons (Figure 15-8) derived from Ivar Jacobson's work.

The objectory process for software engineering involves the following approach:

1. Develop a use case model.
2. Elaborate the use case model into an analysis model.
3. Elaborate the analysis model into a design model.
4. Elaborate the design model into an implementation model.
5. Realize the implementation model.

Because the UML does not dictate a method, this extension only specifies how the UML may be utilized and does not provide more details regarding the above approach.

Table 15-4: Stereotypes for the Objectory Process for Software Engineering

Name	*Application*	*Semantics*
<<use case model>>	Model	The model denotes the topmost package in a use case system.
<<analysis model>>	Model	The model denotes the topmost package in an analysis system.
<<design model>>	Model	The model denotes the topmost package in a design system.

Table 15-4: Stereotypes for the Objectory Process for Software Engineering

Name	*Application*	*Semantics*
<<implementation model>>	Model	The model denotes the topmost package in an implementation system.
<<use case system>>	Package	The package is the top-level package containing use case packages, use cases, actors, and relationships.
<<analysis system>>	Subsystem	The subsystem is the top-level package containing analysis subsystems, analysis service packages, analysis classes, and relationships.
<<design system>>	Subsystem	The subsystem is the top-level package containing design subsystems, design service packages, design classes, and relationships.
<<implementation system>>	Package	The package is the top-level package containing implementation subsystems, components, and relationships.
<<analysis subsystem>>	Subsystem	The subsystem contains other analysis subsystems, analysis service packages, analysis classes, and relationships.
<<design subsystem>>	Subsystem	The subsystem contains other design subsystems, design service packages, design classes, and relationships.
<<implementation subsystem>>	Package	The package contains implementation subsystems, components, and relationships.
<<use case package>>	Package	The package contains use cases, actors, and relationships.
<<analysis service package>>	Subsystem	The package contains analysis classes and relationships.
<<design service package>>	Subsystem	The package contains design classes and relationships.
<<boundary>>	Class	The class defines objects that lie on the periphery of a system and interface with actors outside the system and objects within the system. The class is an analysis class.
<<entity>>	Class	The class defines objects that are passive and do not initiate interactions on their own. The class is an analysis class.

Table 15-4: Stereotypes for the Objectory Process for Software Engineering

Name	*Application*	*Semantics*
<<control>>	Class	The class defines objects that control interactions among a collection of objects. The class usually has behavior specific to one use case. The class is an analysis class.
<<communicates>>	Association	The association denotes a relationship between actors and use cases. The relationship specifies that the actor sends messages to the use case, and the use case sends a message to the actor. One-way and two-way navigation capture the direction of the communication. The stereotype may be omitted since it is the only kind of relationship between actors and use cases.
<<subscribes>>	Association	The association denotes a relationship between a source subscriber class and a target publisher class. The subscriber specifies a set of events, and the publisher notifies the subscriber when one of the events occurs in the publisher.
<<use case realization>>	Collaboration	The collaboration realizes or implements a use case.

Table 15-5: Association Rules for the Objectory Process for Software Engineering

Association	*Source*	*Possible Targets*
<<communicates>>	Actor	Boundary
	Boundary	Actor Boundary Control Entity
	Control	Boundary Control Entity
	Entity	Entity
<<subscribes>>	Boundary	Entity
	Entity	Entity

UML Extension for Business Modeling

The UML Extension for Business Modeling document registers a domain-specific extension to the UML. It includes a set of stereotypes (Table 15-6), various association rules (Table 15-7), and a set of stereotype icons (Figure 15-9) derived from Ivar Jacobson's work.

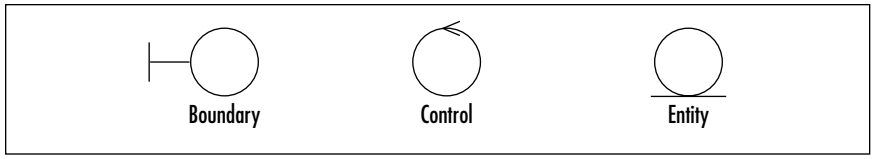


Figure 15-8: Stereotype Icons for the Objectory Process for Software Engineering

Business systems may be characterized by internal models and external models. Internal models are object models that describe the things internal to the business. External models are use case models that describe business processes involved in interactions with external parties.

Table 15-6: Business Modeling Stereotypes

Name	*Application*	*Semantics*
<<use case model>>	Model	The model denotes the business processes of a business and interactions with external parties. The model describes business processes as use cases, parties external to the business as actors, and relationships between external parties and business processes.
<<use case system>>	Package	The package is the top-level package containing use case packages, use cases, actors, and relationships.
<<use case package>>	Package	The package contains use cases, actors, and relationships.
<<object model>>	Model	The model denotes the topmost package in an object system. The model describes things internal to the business system.
<<object system>>	Subsystem	The subsystem is the top-level subsystem in an object model containing organization units, classes (workers, work units, and entities), and relationships.
<<organization unit>>	Subsystem	The subsystem is an organization unit of the actual business consisting of organization units, work units, classes (workers and entities), and relationships.
<<work unit>>	Subsystem	The subsystem contains one or more entities. The objects form a task-oriented view to end users.

Table 15-6: Business Modeling Stereotypes (continued)

Name	*Application*	*Semantics*
<<worker>>	Class	The class defines a human that acts within the system. Workers interact and manipulate entities while participating in use case realizations.
<<case worker>>	Class	The class defines a worker who interacts directly with actors outside a system.
<<internal worker>>	Class	The class defines a worker who interacts with other workers and entities inside a system.
<<entity>>	Class	The class defines objects that are passive and do not initiate interactions on their own. These classes provide the basis for sharing among workers involved in interactions.
<<communicates>>	Association	The association denotes a relationship between two instances that interact. The relationship specifies that the instances interact by sending and receiving messages. One-way and two-way navigation capture the direction of the communication.
<<subscribes>>	Association	The association denotes a relationship between a source subscriber class and a target publisher class. The subscriber specifies a set of events, and the publisher notifies the subscriber when one of the events occurs in the publisher.
<<use case realization>>	Collaboration	The collaboration realizes or implements a use case.

Table 15-7: Business Modeling Association Rules

Association	*Source*	*Possible Targets*
<<communicates>>	Actor	Case worker Work unit
	Case worker	Actor Case worker Entity Internal worker Work unit
	Entity	Entity Work unit

Table 15-7: Business Modeling Association Rules

Association	*Source*	*Possible Targets*
	Internal worker	Case worker Entity Internal worker Work unit
	Work unit	Actor Case worker Entity Internal worker Work unit
<<subscribes>>	Entity	Entity
	Internal worker	Entity Work unit
	Work unit	Entity Work unit

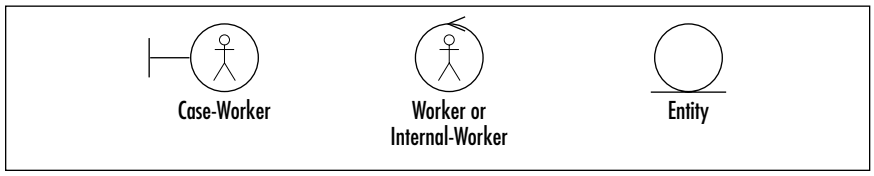


Figure 15-9: Business Modeling Stereotype Icons