MagicDraw CSV Import Plug-in

Version 17.0.2 SP5

User Guide

No Magic, Inc.

2013

Table of Contents

Introduction

CSV Import is a MagicDraw plug-in that will read values in a comma separated values file and create Model elements, diagrams and relationships from that data. The plug-in uses the CSVReader application to parse the CSV file. You may view the requirements for this application at http://www.csvreader.com.

What is a Comma Separated Values File?

Imagine information laid out in the style of a spreadsheet. Each row contains information about a distinct type of thing; each column contains an attribute of that thing. For example:

First Name	Last Name	City	State	Date Of Birth
John	Wilson	Miami	FL	04-22-1968
Mary	Kelley	Newark	NJ	07-18-1977

As you can see, each row contains information about a specific person and each column contains discreet information *about* that person. A CSV file is a text file containing this information, but a comma separates each column. It might look like this:

```
John,Wilson,Miami,FL,04-22-1968

Mary,Kelley,Newark,NJ,07-18-1977
```

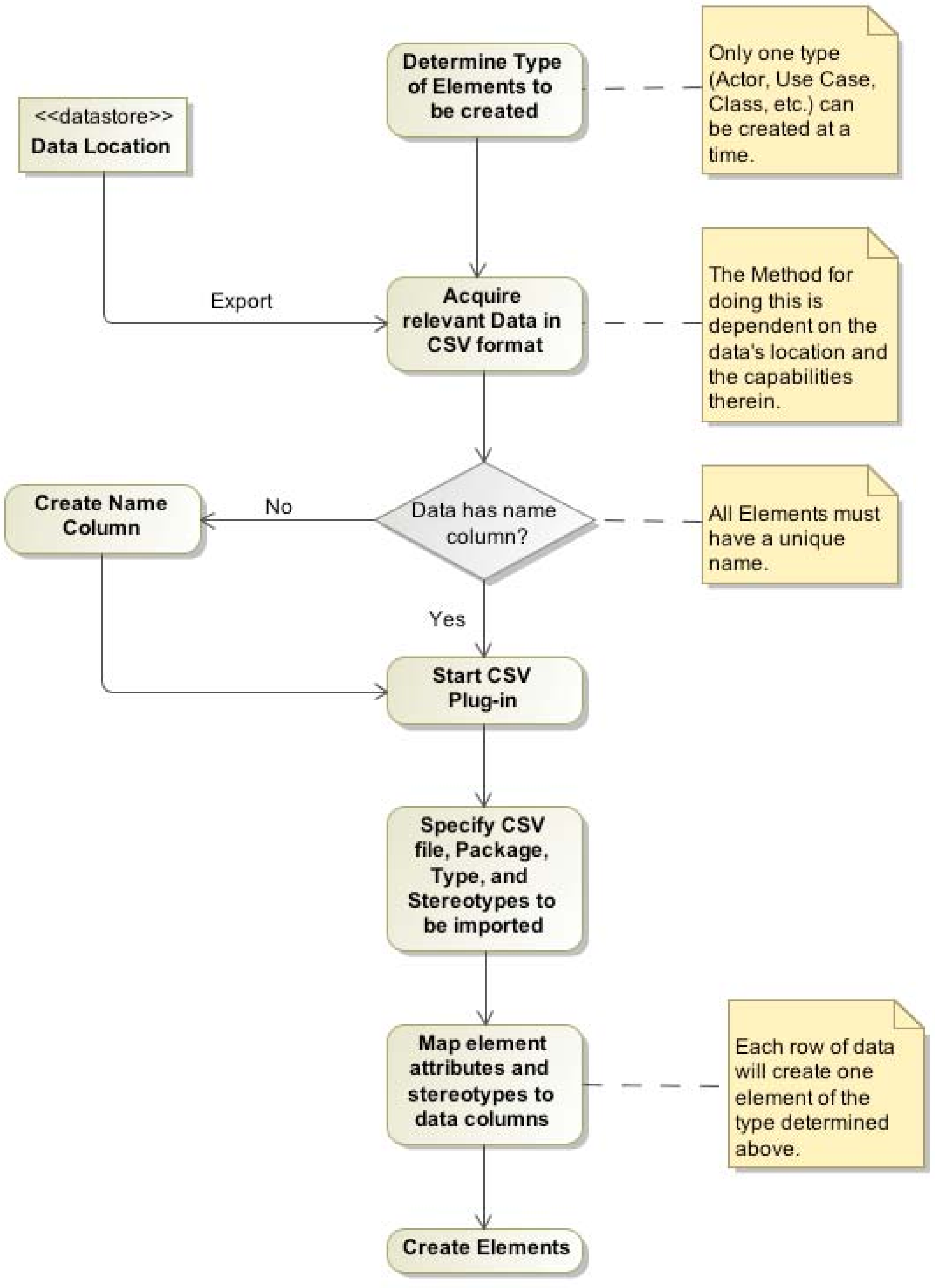
How Do I Get My Data into Comma Separated Values format?

This depends on where your information is to begin with. If it is in Microsoft Excel or some other spreadsheet program, there is usually a way to export or save the data as CSV. If it is in an SQL database, you may have to write a query or use a data access tool to get the data into the proper format.

Before You Begin

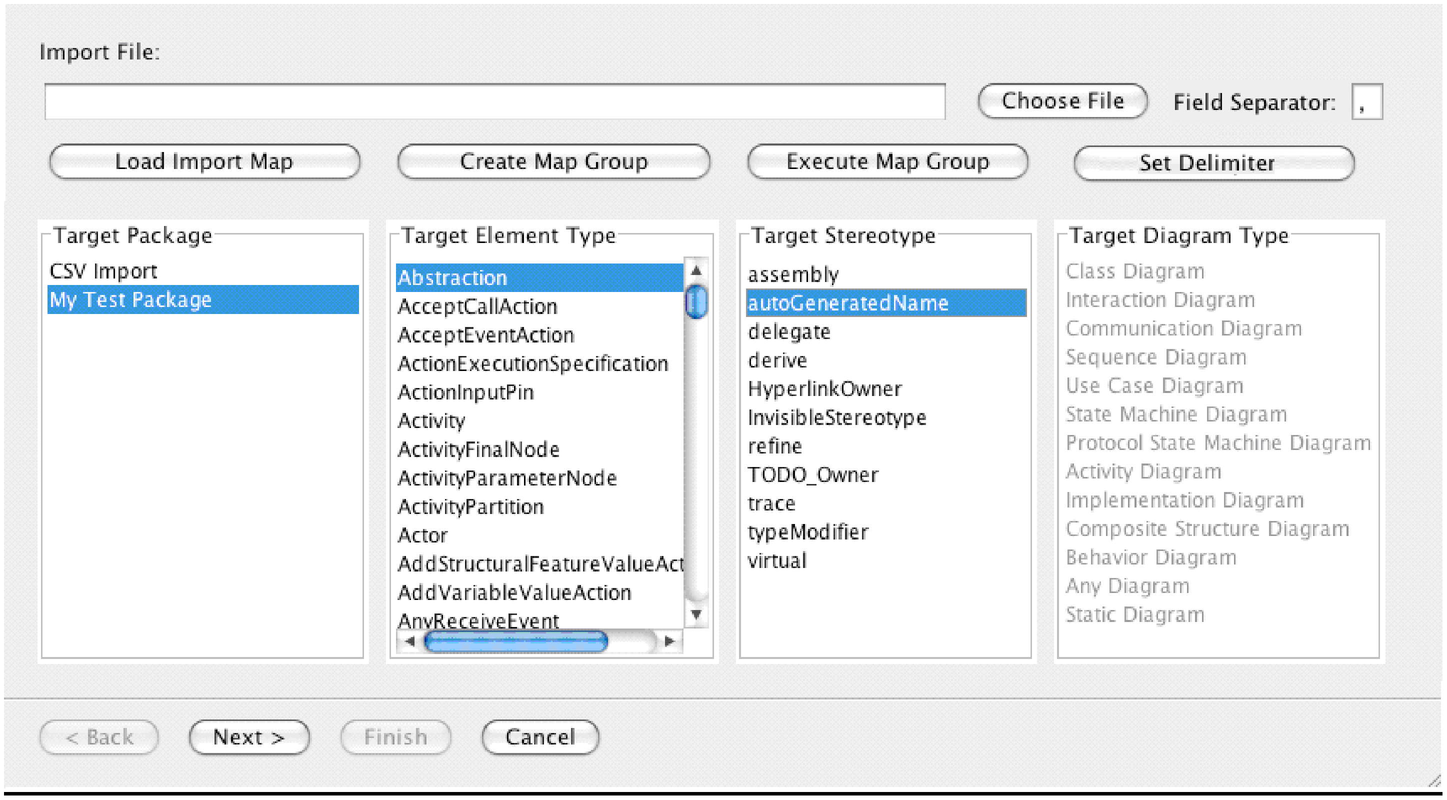
You should be familiar with the MagicDraw program and how the various diagrams, model elements, and relationships hang together. This plug-in assumes you are and will not validate the things you are trying to do before attempting to do them. It is therefore your responsibility to import clean data in the appropriate order. In general, you import diagrams first, then model elements, then relationships.

- The plug-in will not allow you to create a root package within MagicDraw. Therefore, you must manually create any root packages in which you want to create elements, diagrams, and relationships. You may use the plug-in to create sub packages.
- The plug-in avoids duplication by allowing you to choose the Key property. Therefore, on every import, the plug-in can update that element if it already exists in your model and create it if it does not. The key property identify the uniqueness of elements, so you can now have a class named Class1 with a Property element named XYZ and another class named Class2 with a Property element named XYZ. Any elements that you wish the importer to place on a diagram should have unique names at the package level since there is not a way to establish the ownership of an element and the diagram on which to place it at the same time.
- Before using this plug-in to create elements, you should create them by hand in a test model to ensure that you know if any other prior elements are required by the model. You can get more on this in the Examples section under Prototyping Element in a Test Model Before You Begin. To be on the safe side, always make a backup of your model before running this plug-in.



Determine Type of Elements to be created
Only one type (Actor, Use Case, Class, etc.) can be created at a time.
<<datastore>> Data Location
Export
Acquire relevant Data in CSV format
The Method for doing this is dependent on the data's location and the capabilities therein.
Create Name Column
No
Data has name column?
All Elements must have a unique name.
Yes
Start CSV Plug-in
Specify CSV file, Package, Type, and Stereotypes to be imported
Map element attributes and stereotypes to data columns
Each row of data will create one element of the type determined above.
Create Elements

Figure 1 - The basic process for using this plug-in


Import File:

Choose File Field Separator: ,

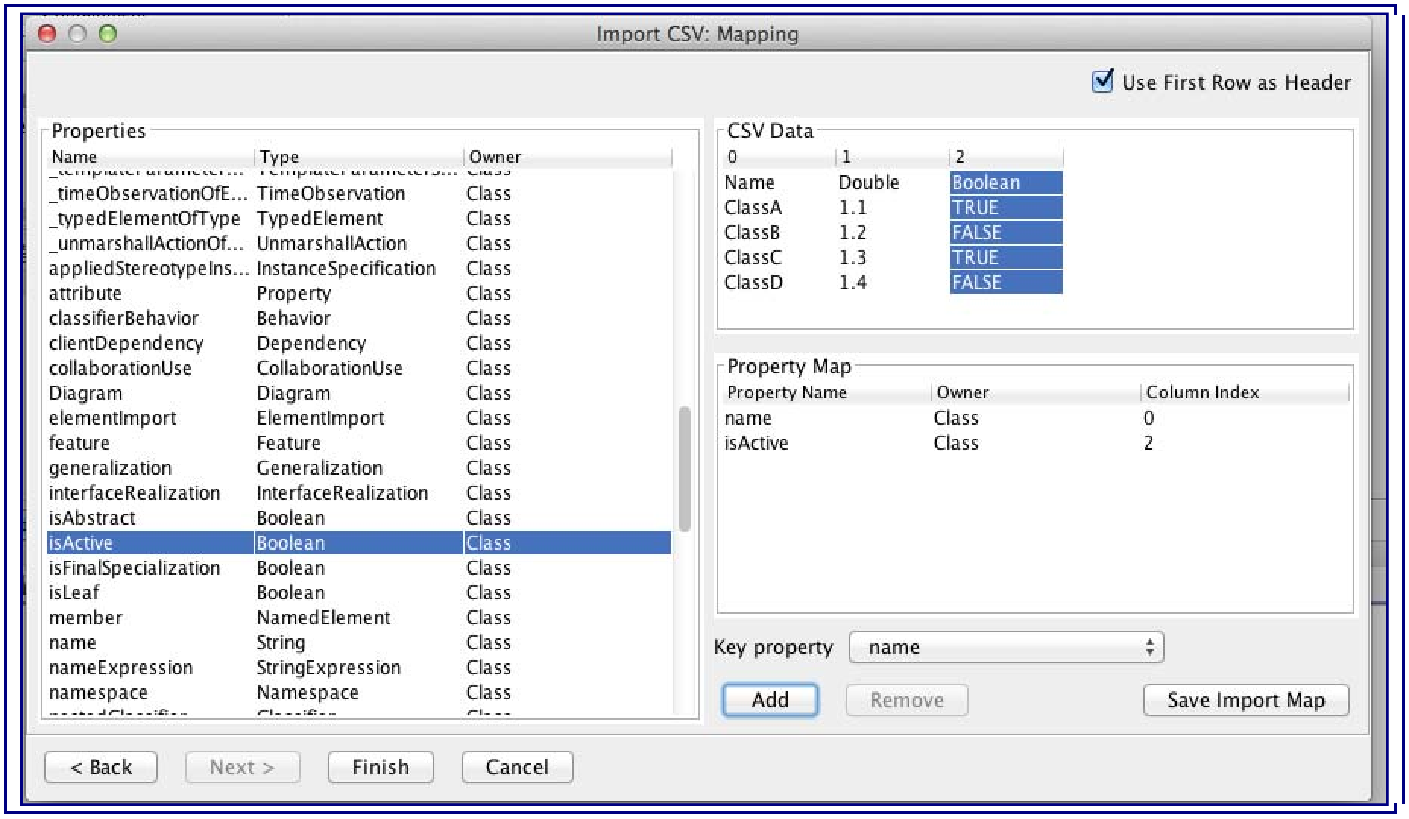
Load Import Map | Create Map Group | Execute Map Group | Set Delimiter

Target Package	Target Element Type	Target Stereotype	Target Diagram Type
CSV Import	Abstraction	assembly	Class Diagram
My Test Package	AcceptCallAction	autoGeneratedName	Interaction Diagram
	AcceptEventAction	delegate	Communication Diagram
	ActionExecutionSpecification	derive	Sequence Diagram
	ActionInputPin	HyperlinkOwner	Use Case Diagram
	Activity	InvisibleStereotype	State Machine Diagram
	ActivityFinalNode	refine	Protocol State Machine Diagram
	ActivityParameterNode	TODO_Owner	Activity Diagram
	ActivityPartition	trace	Implementation Diagram
	Actor	typeModifier	Composite Structure Diagram
	AddStructuralFeatureValueAct	virtual	Behavior Diagram
	AddVariableValueAction		Any Diagram
	AnyReceiveEvent		Static Diagram

< Back | Next > | Finish | Cancel

The first screen is known as the Setup screen. Here, you select the CSV file you wish to import and select which package you are importing into, the model Element Types and Stereotypes you are importing. If you use other character to split each column other than "," (comma), you can specify in the Field Separator. When you are done, press next to move to the following screen. If you select the Diagram element type, the diagram type list will be enabled and you can select the type of diagram you wish to create.

Note: While you can use the same file to import multiple types of model elements, the plug-in will only allow one type to be imported at a time. The same rule applies to diagram types.


Import CSV: Mapping

☑ Use First Row as Header

Properties

Name	Type	Owner
_timeObservationOfE...	TimeObservation	Class
_typedElementOfType	TypedElement	Class
_unmarshallActionOf...	UnmarshallAction	Class
appliedStereotypeIns...	InstanceSpecification	Class
attribute	Property	Class
classifierBehavior	Behavior	Class
clientDependency	Dependency	Class
collaborationUse	CollaborationUse	Class
Diagram	Diagram	Class
elementImport	ElementImport	Class
feature	Feature	Class
generalization	Generalization	Class
interfaceRealization	InterfaceRealization	Class
isAbstract	Boolean	Class
isActive	Boolean	Class
isFinalSpecialization	Boolean	Class
isLeaf	Boolean	Class
member	NamedElement	Class
name	String	Class
nameExpression	StringExpression	Class
namespace	Namespace	Class

CSV Data

0	1	2
Name	Double	Boolean
ClassA	1.1	TRUE
ClassB	1.2	FALSE
ClassC	1.3	TRUE
ClassD	1.4	FALSE

Property Map

Property Name	Owner	Column Index
name	Class	0
isActive	Class	2

Key property: name

The second screen is known as the mappings screen. This screen is where you will tell the plug-in whether or not to use the first row of your file as headers, and what column of data maps to which attribute. To map an attribute, select the column from your CSV data and a row from the properties table, and then press the Add button. The association between the attribute and the column in the CSV file will be represented in the Property Maps table. If you want to remove one of these associations, select its row in the Property Maps table and press the Remove button. Once the mapping is done, you can choose the Key property to identify the uniqueness of importing data. So, the plug-in can update that element if it already exists in your model and create it if it does not.

Press Finish on this screen to start the import. The MagicDraw log area will contain messages concerning the data being imported.

Mapping Attributes

Data Conversions

The plug-in is designed to convert textual data into model elements and attributes. You may be familiar with names like Integer or Decimal and know that those are numbers. Boolean is a True/False attribute, and String is textual data. You will also see a lot of internal MagicDraw data types like Element or Component. In those cases, the plug-in will assume that what you have specified is the item's name. It will then go looking for an element with that name. Be sure you know what you're doing. If you attempt to assign an element of one type to another type, errors will occur. The importer will display all attributes, whether it knows how to convert them or not. It is up to you to provide a value that the importer can understand. Mostly, you will simply need to provide the element's name. Moreover you can set the Type of a property. Just specify the name of the type as you see it in the Type drop down list in an element's specification. The spelling is case sensitive.

The Owner Attribute

The Owner attribute can be mapped like any other attribute. If you do not map it, then the package you have selected becomes the owner of the element you are creating. Since the package is usually the owner of all the elements, you should not need to map this very often. There are a couple of exceptions to this rule:

- If you map the name of a diagram to the owner attribute, the plug-in will assume that you want to place that element on the diagram. If the element is new, it will be given the package as its owner and placed on the diagram.

Key property

The importer provides you a flexible way to identify the uniqueness of elements. You can either choose Name or other attribute. Thus, instead of creating the duplicate elements, the existing element will get updated when you import data.

Adding Importing Elements to Diagrams

To show the elements you are importing on a diagram, map the Diagram property with the column that represents the diagram.

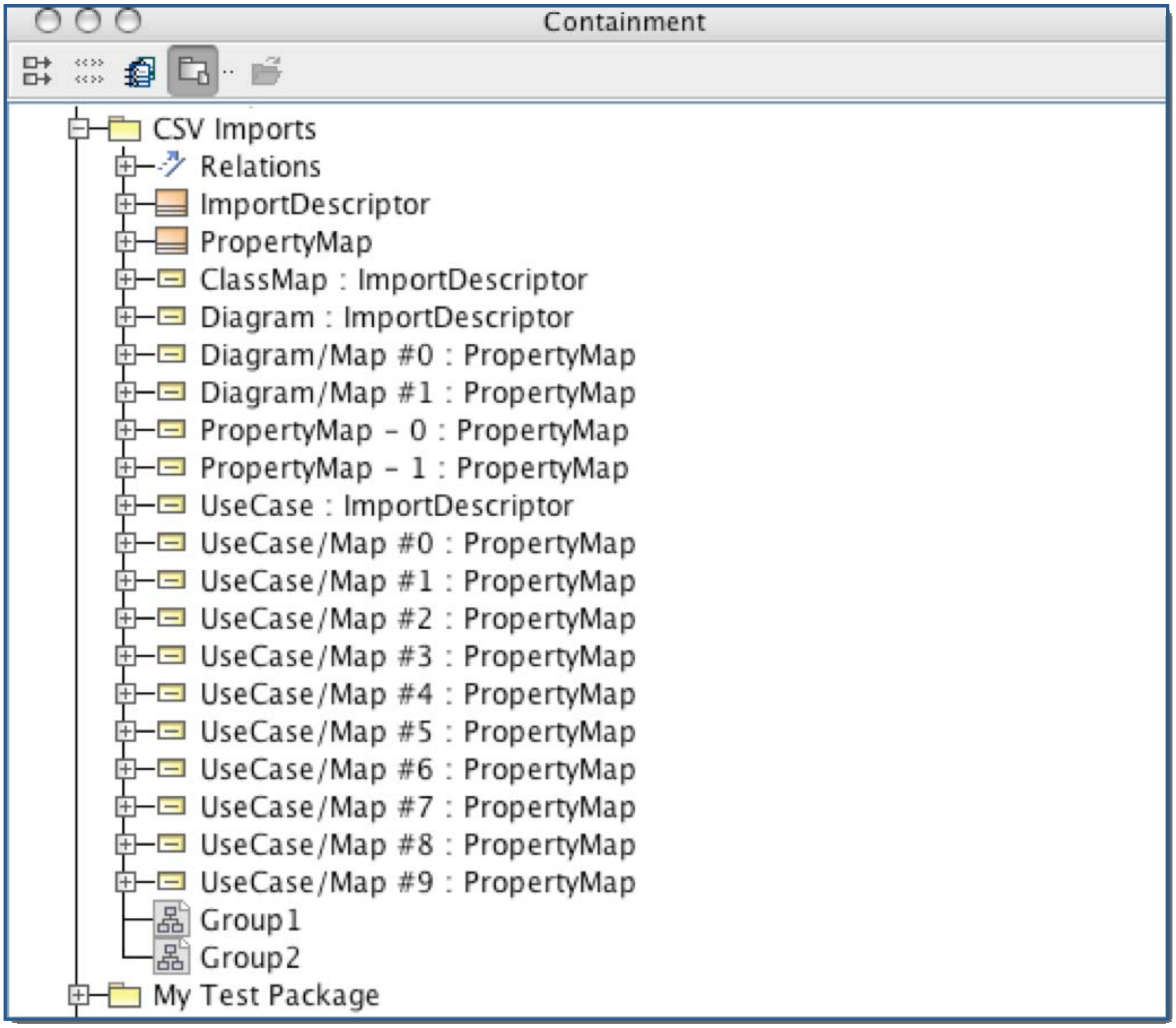
Creating Relationships

When creating Relationships, the plug-in will ask you to provide the source and target. Map these to the names of the existing model elements you want to connect. These elements should be on the same diagram. If they are not, the plug-in will not report an error. However, when you go back and look at your diagram, you will see a line drawn to nowhere. If you have a model element that appears on multiple diagrams, you will need to map the diagram name to the Owner attribute of the Relationship to ensure the relationship is drawn on the correct diagram. Make this mapping **after the name and before the source and target** mappings in the Property Maps Table. This is the only instance where the order of the mappings is significant.

Saving Your Choices

Once you have the setup and mappings that you want, you may wish to perform this import multiple times or share it with other users. You may also wish to create groups of imports that will be done in sequence. The plug-in offers you to save this information in the MagicDraw model.

The first time you save something to the model, the plug-in will create a package in the model called "CSV Imports". This package is open to you and can be edited at will. The only circumstance under which you should edit anything in this package is to delete items you no longer want.

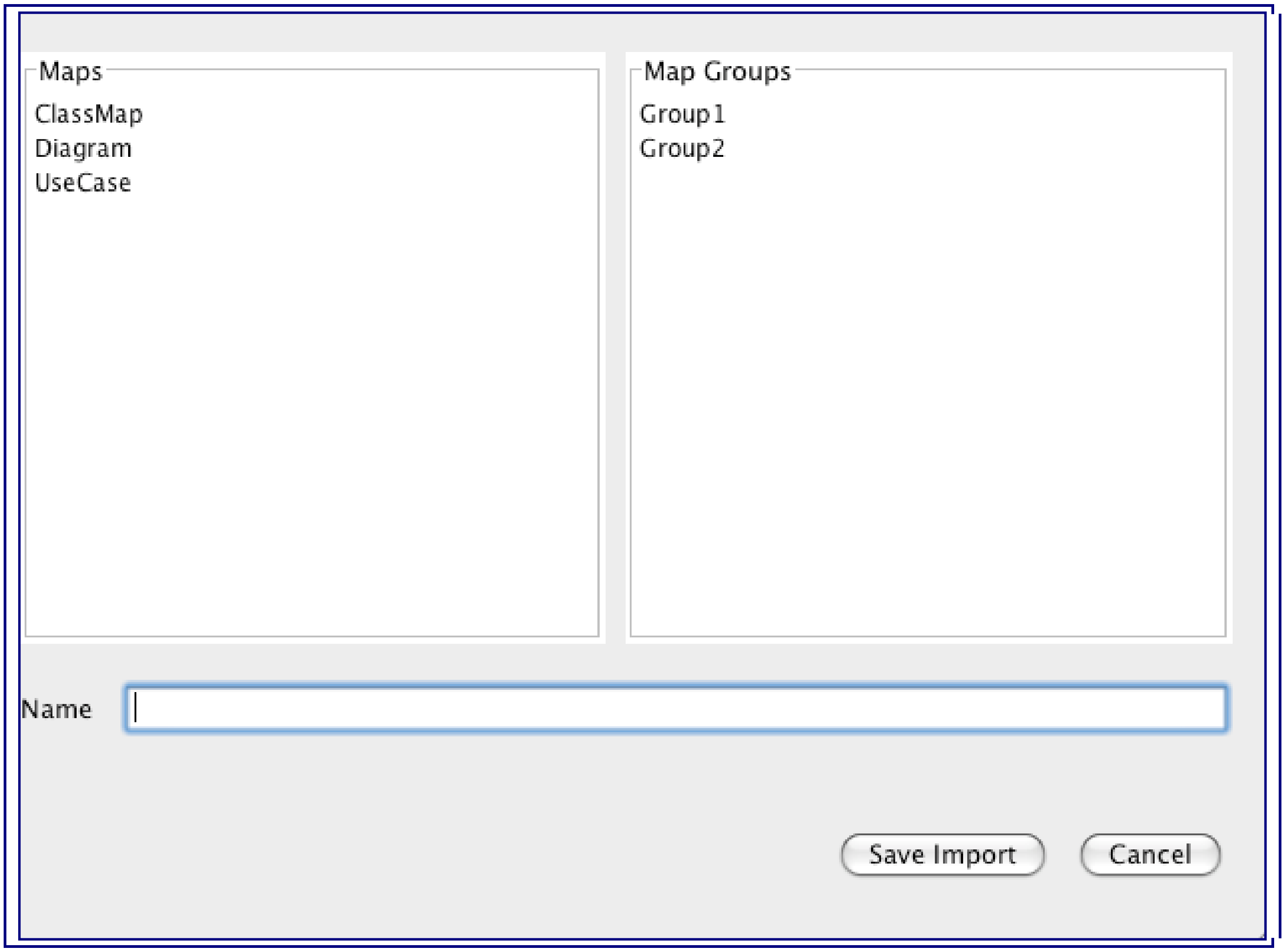

Containment
CSV Imports
Relations
ImportDescriptor
PropertyMap
ClassMap : ImportDescriptor
Diagram : ImportDescriptor
Diagram/Map #0 : PropertyMap
Diagram/Map #1 : PropertyMap
PropertyMap - 0 : PropertyMap
PropertyMap - 1 : PropertyMap
UseCase : ImportDescriptor
UseCase/Map #0 : PropertyMap
UseCase/Map #1 : PropertyMap
UseCase/Map #2 : PropertyMap
UseCase/Map #3 : PropertyMap
UseCase/Map #4 : PropertyMap
UseCase/Map #5 : PropertyMap
UseCase/Map #6 : PropertyMap
UseCase/Map #7 : PropertyMap
UseCase/Map #8 : PropertyMap
UseCase/Map #9 : PropertyMap
Group1
Group2
My Test Package

Import maps are stored as Instances of the ImportDescriptor class along with a set of instances of the PropertyMap class. If you want to delete a map from your model, delete all the instances. **Do not delete the ImportDescriptor or PropertyMap classes themselves.** Map groups are stored as class diagrams. To delete a map group, open the diagram and manually delete all the Dependencies between the instances on the diagram. Then, close the diagram and delete it from the containment area. Then delete any maps that you no longer want.

If you accidentally delete the ImportDescriptor or PropertyMap classes, they will be recreated the next time you save something to the Model.

Saving Maps

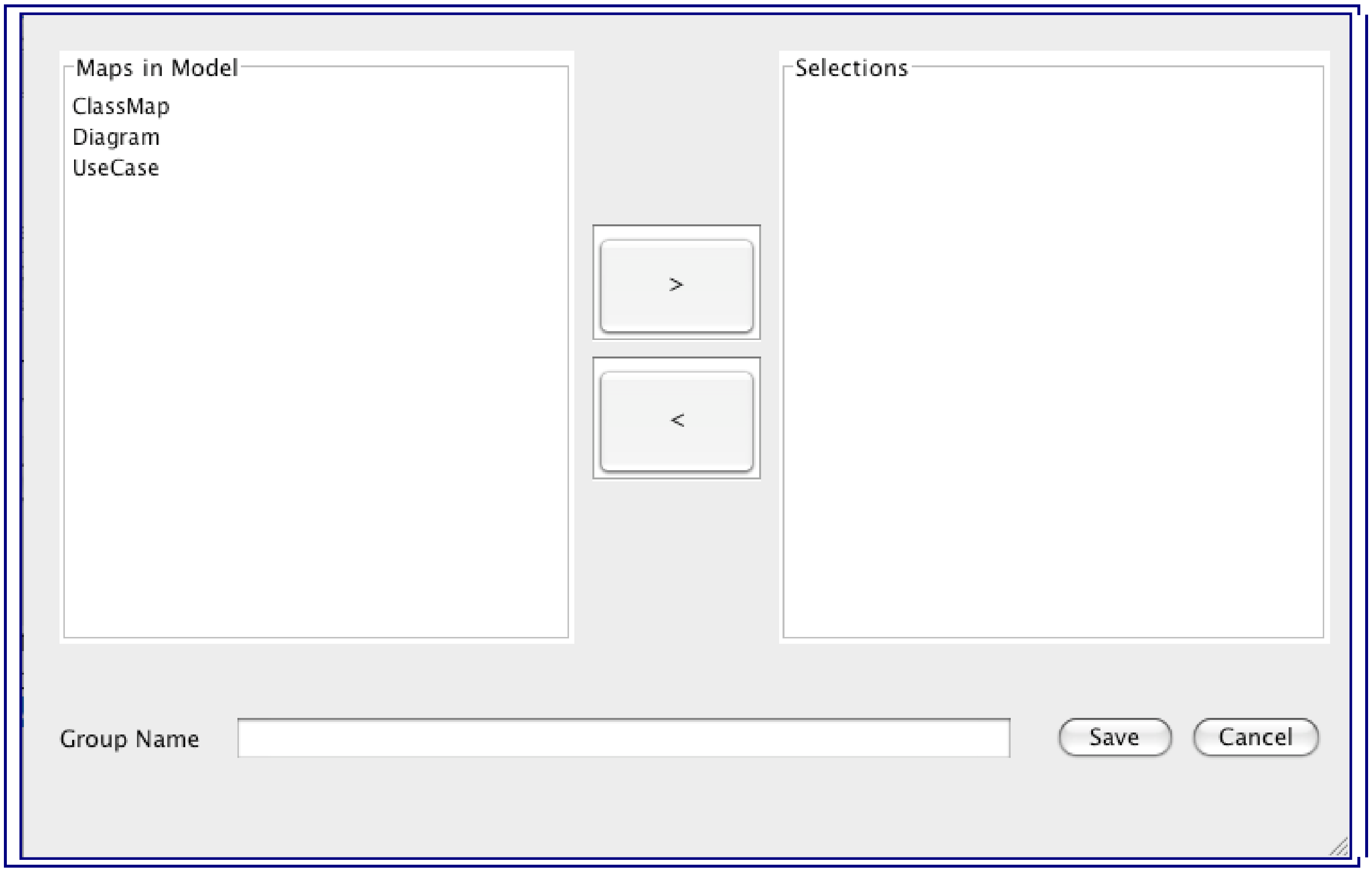
If you press the Save Map button, a window will pop up asking you for the name you want to save the map under.



The list on the left shows the names of the maps already in your model. If you select one of these names, it will be copied to the Name field. If you do not change it there, it will be overwritten. You will also see this screen when you are loading maps from your model.

Saving Map Groups

When you choose to create a map group, a window will pop up asking you which model maps you want to add to the group.



The list on the left shows the maps already created in the model. You may select them individually or in groups to move to the list on the right with the **>** button. These maps will be imported in the sequence you specify. If you make a mistake, you can use the **<** button to remove a selection.

Press the Execute group to run these map groups as a batch of maps. In File mode, you'll be asked for the file. In Model mode, you'll be shown the same screen as seen in "Saving Maps."

You may also use the Model and File modes interchangeably once during a session. So, you could use File mode to develop your map and then switch to Model mode to store it for others to use.

Creating DoDAF And Other Custom Model Elements

Creating custom model element such as DoDAF diagrams and elements is not that much different than creating normal UML items and mostly entails simply applying stereotypes to the elements you're creating. Make sure you have loaded the custom profile into your model, and proceed from there.

Diagrams: If you want to create an OV-2 diagram for example, use the instructions from the section "Creating and Populating Class Diagrams" to import a Class Diagram, but be sure to select the OV-2 Diagram Type.

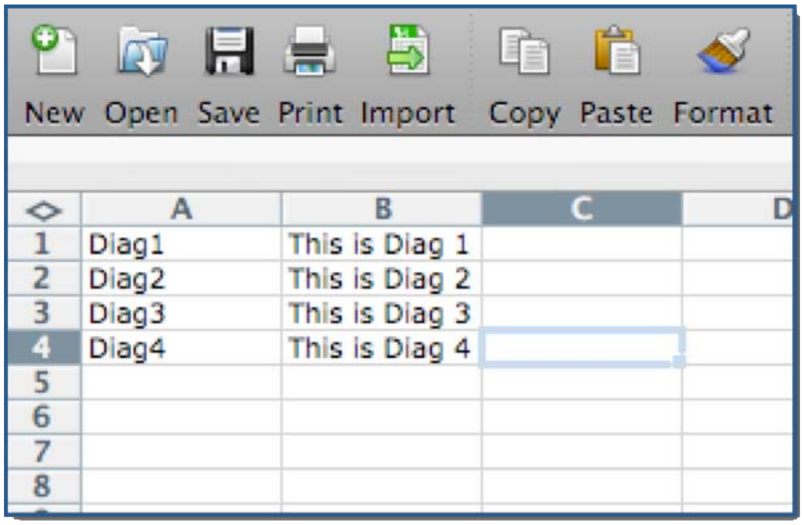
Operational Nodes: If you import elements of type Class and apply the Operational Node stereotype, those will be created in your model and placed on your diagrams if you so choose.

Needlines: A Needline is just a Relationship element with the Needline stereotype applied. You may create them the same way you create Relationships.

Examples

Creating and Populating Class Diagrams

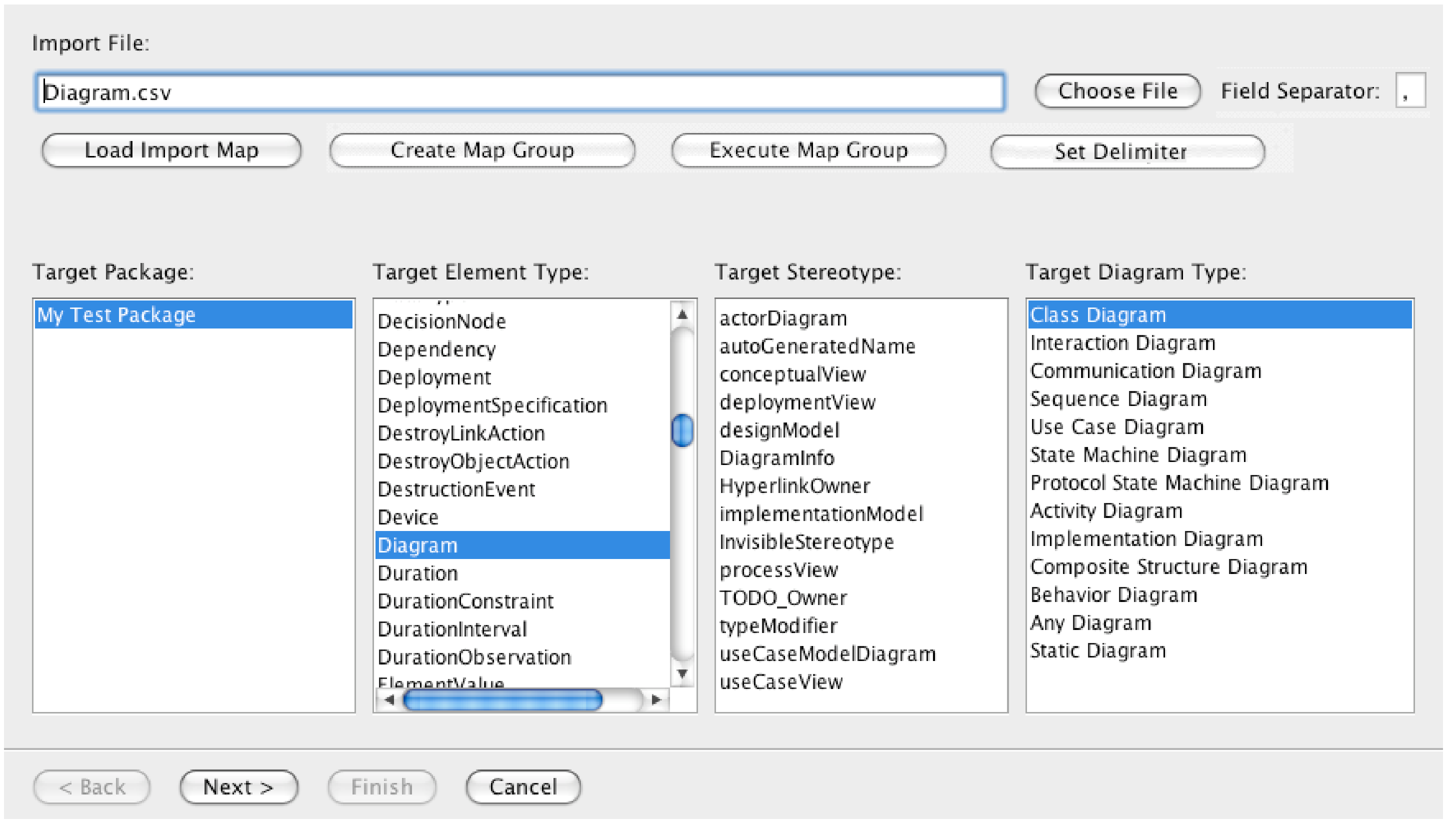
Start with an Excel spreadsheet:



	A	B	C	D
1	Diag1	This is Diag 1		
2	Diag2	This is Diag 2		
3	Diag3	This is Diag 3		
4	Diag4	This is Diag 4		
5				
6				
7				
8				

Export this to CSV Format.

Now, bring up the CSV Import Plug-in.


Import File: Diagram.csv

Field Separator: ,

Target Package:

My Test Package

Target Element Type:

DecisionNode
Dependency
Deployment
DeploymentSpecification
DestroyLinkAction
DestroyObjectAction
DestructionEvent
Device
Diagram
Duration
DurationConstraint
DurationInterval
DurationObservation
ElementValue

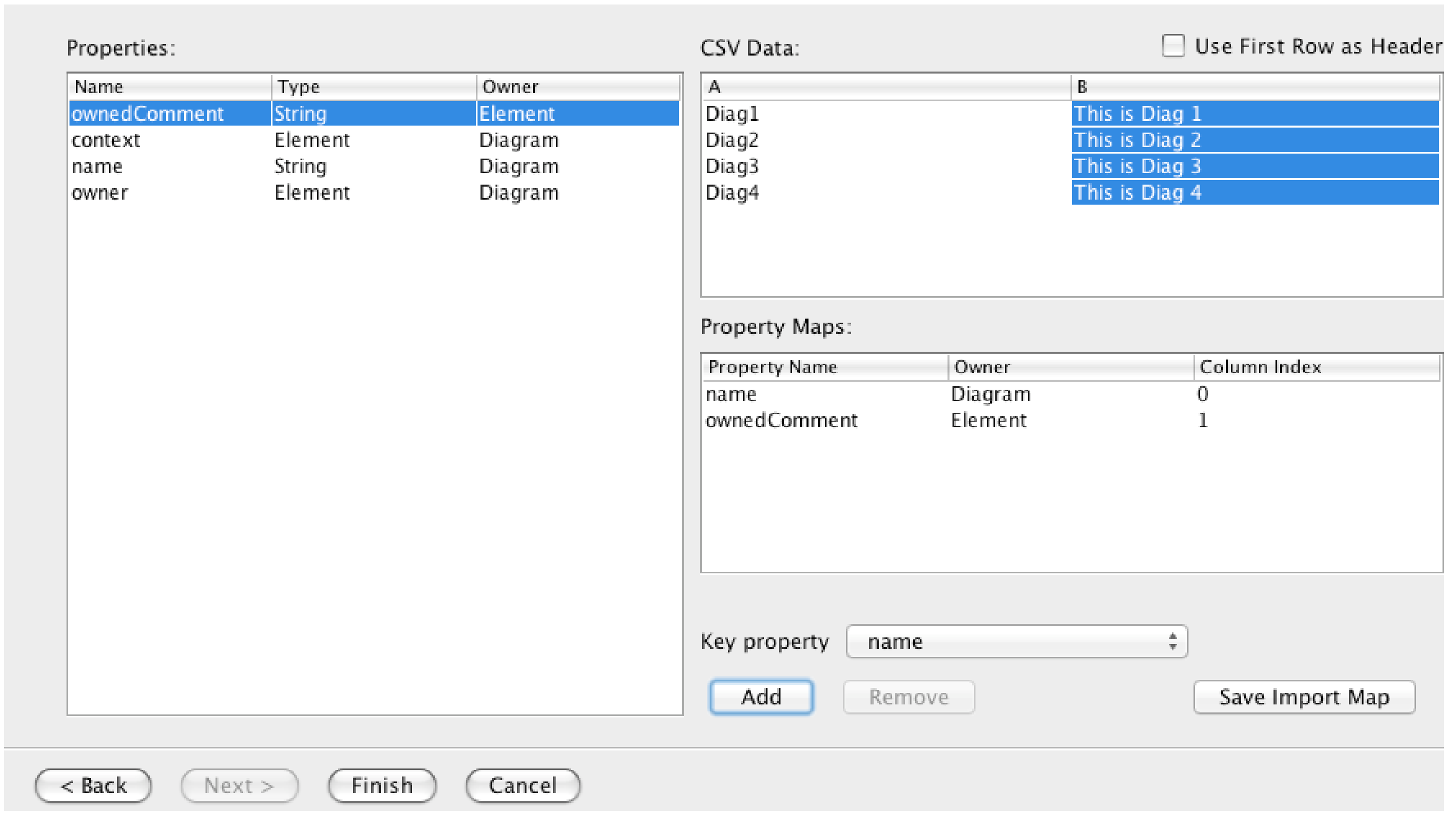
Target Stereotype:

actorDiagram
autoGeneratedName
conceptualView
deploymentView
designModel
DiagramInfo
HyperlinkOwner
implementationModel
InvisibleStereotype
processView
TODO_Owner
typeModifier
useCaseModelDiagram
useCaseView

Target Diagram Type:

Class Diagram
Interaction Diagram
Communication Diagram
Sequence Diagram
Use Case Diagram
State Machine Diagram
Protocol State Machine Diagram
Activity Diagram
Implementation Diagram
Composite Structure Diagram
Behavior Diagram
Any Diagram
Static Diagram

Press Next to move to the next screen.


Properties:

Name	Type	Owner
ownedComment	String	Element
context	Element	Diagram
name	String	Diagram
owner	Element	Diagram

CSV Data:

Use First Row as Header

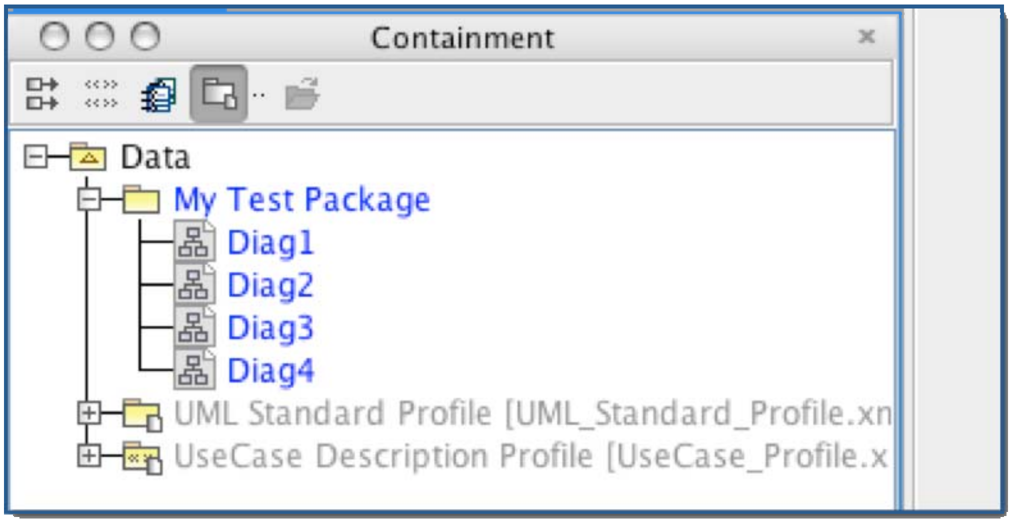
A	B
Diag1	This is Diag 1
Diag2	This is Diag 2
Diag3	This is Diag 3
Diag4	This is Diag 4

Property Maps:

Property Name	Owner	Column Index
name	Diagram	0
ownedComment	Element	1

Key property: name

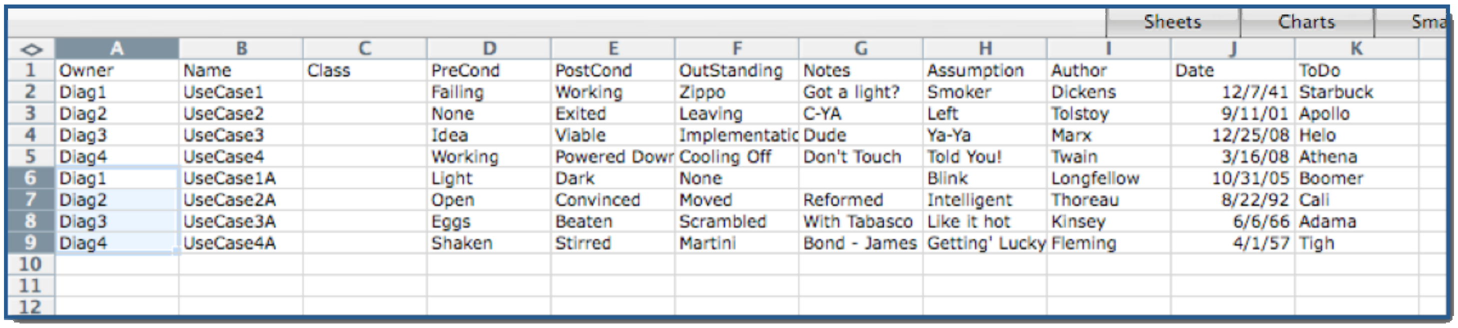
Press Finish and look at your MagicDraw containment window.



- Data
  - My Test Package
    - Diag1
    - Diag2
    - Diag3
    - Diag4
  - UML Standard Profile [UML_Standard_Profile.xn
  - UseCase Description Profile [UseCase_Profile.x

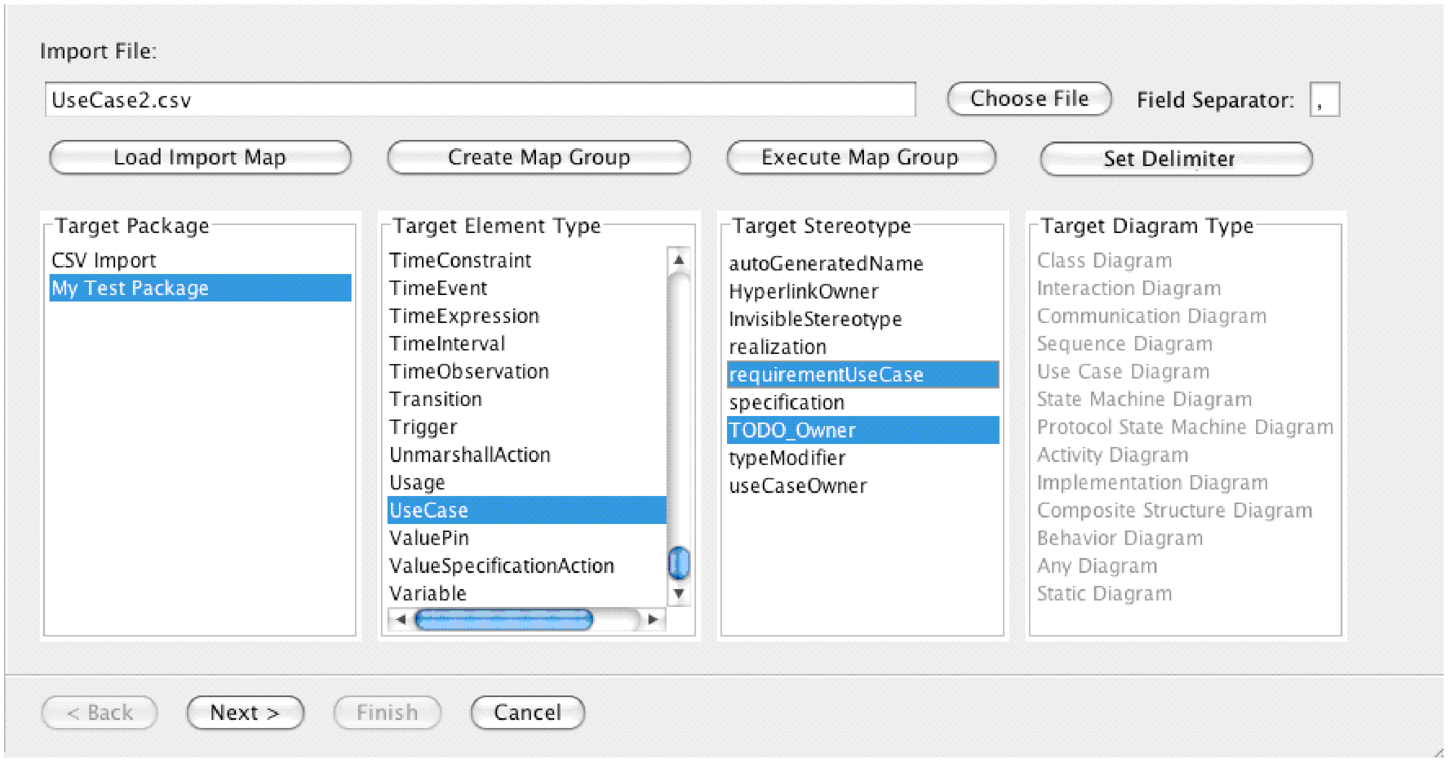
Creating Classes into Diagram

Now, let's add some elements to these diagrams. Again, we'll start out with spreadsheet data:


Sheets
Charts
Sma

	A	B	C	D	E	F	G	H	I	J	K
1	Owner	Name	Class	PreCond	PostCond	OutStanding	Notes	Assumption	Author	Date	ToDo
2	Diag1	UseCase1		Failing	Working	Zippo	Got a light?	Smoker	Dickens	12/7/41	Starbuck
3	Diag2	UseCase2		None	Exited	Leaving	C-YA	Left	Tolstoy	9/11/01	Apollo
4	Diag3	UseCase3		Idea	Viable	Implementatic	Dude	Ya-Ya	Marx	12/25/08	Helo
5	Diag4	UseCase4		Working	Powered Dowr	Cooling Off	Don't Touch	Told You!	Twain	3/16/08	Athena
6	Diag1	UseCase1A		Light	Dark	None		Blink	Longfellow	10/31/05	Boomer
7	Diag2	UseCase2A		Open	Convinced	Moved	Reformed	Intelligent	Thoreau	8/22/92	Cali
8	Diag3	UseCase3A		Eggs	Beaten	Scrambled	With Tabasco	Like it hot	Kinsey	6/6/66	Adama
9	Diag4	UseCase4A		Shaken	Stirred	Martini	Bond - James	Getting' Lucky	Fleming	4/1/57	Tigh
10											
11											
12											

Then, we run the import plug-in.


Import File: UseCase2.csv

Field Separator: ,

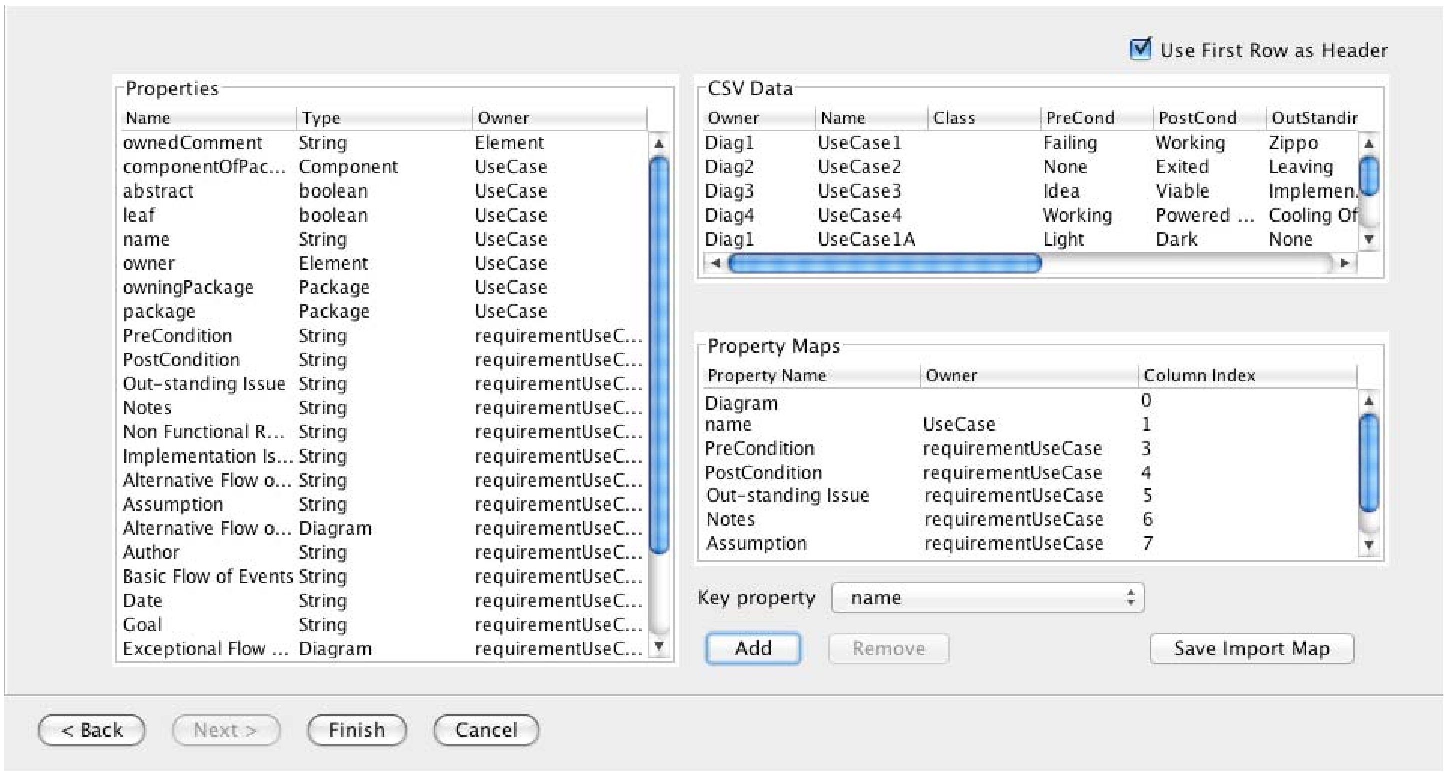
Target Package
- CSV Import
- My Test Package (selected)

Target Element Type
- TimeConstraint
- TimeEvent
- TimeExpression
- TimeInterval
- TimeObservation
- Transition
- Trigger
- UnmarshallAction
- Usage
- UseCase (selected)
- ValuePin
- ValueSpecificationAction
- Variable

Target Stereotype
- autoGeneratedName
- HyperlinkOwner
- InvisibleStereotype
- realization
- requirementUseCase (selected)
- specification
- TODO_Owner (selected)
- typeModifier
- useCaseOwner

Target Diagram Type
- Class Diagram
- Interaction Diagram
- Communication Diagram
- Sequence Diagram
- Use Case Diagram
- State Machine Diagram
- Protocol State Machine Diagram
- Activity Diagram
- Implementation Diagram
- Composite Structure Diagram
- Behavior Diagram
- Any Diagram
- Static Diagram

Press Next to move to the next screen.


☑ Use First Row as Header

Properties

Name	Type	Owner
ownedComment	String	Element
componentOfPac...	Component	UseCase
abstract	boolean	UseCase
leaf	boolean	UseCase
name	String	UseCase
owner	Element	UseCase
owningPackage	Package	UseCase
package	Package	UseCase
PreCondition	String	requirementUseC...
PostCondition	String	requirementUseC...
Out-standing Issue	String	requirementUseC...
Notes	String	requirementUseC...
Non Functional R...	String	requirementUseC...
Implementation Is...	String	requirementUseC...
Alternative Flow o...	String	requirementUseC...
Assumption	String	requirementUseC...
Alternative Flow o...	Diagram	requirementUseC...
Author	String	requirementUseC...
Basic Flow of Events	String	requirementUseC...
Date	String	requirementUseC...
Goal	String	requirementUseC...
Exceptional Flow ...	Diagram	requirementUseC...

CSV Data

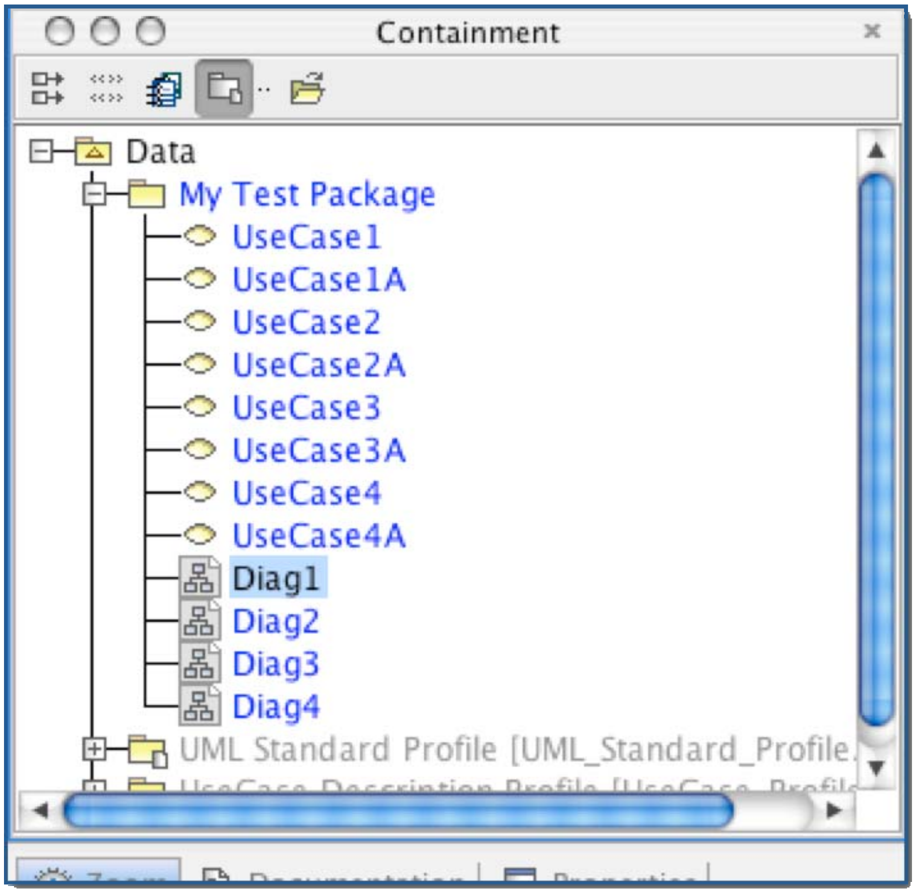
Owner	Name	Class	PreCond	PostCond	OutStandir
Diag1	UseCase1		Failing	Working	Zippo
Diag2	UseCase2		None	Exited	Leaving
Diag3	UseCase3		Idea	Viable	Implemen.
Diag4	UseCase4		Working	Powered ...	Cooling Of
Diag1	UseCase1A		Light	Dark	None

Property Maps

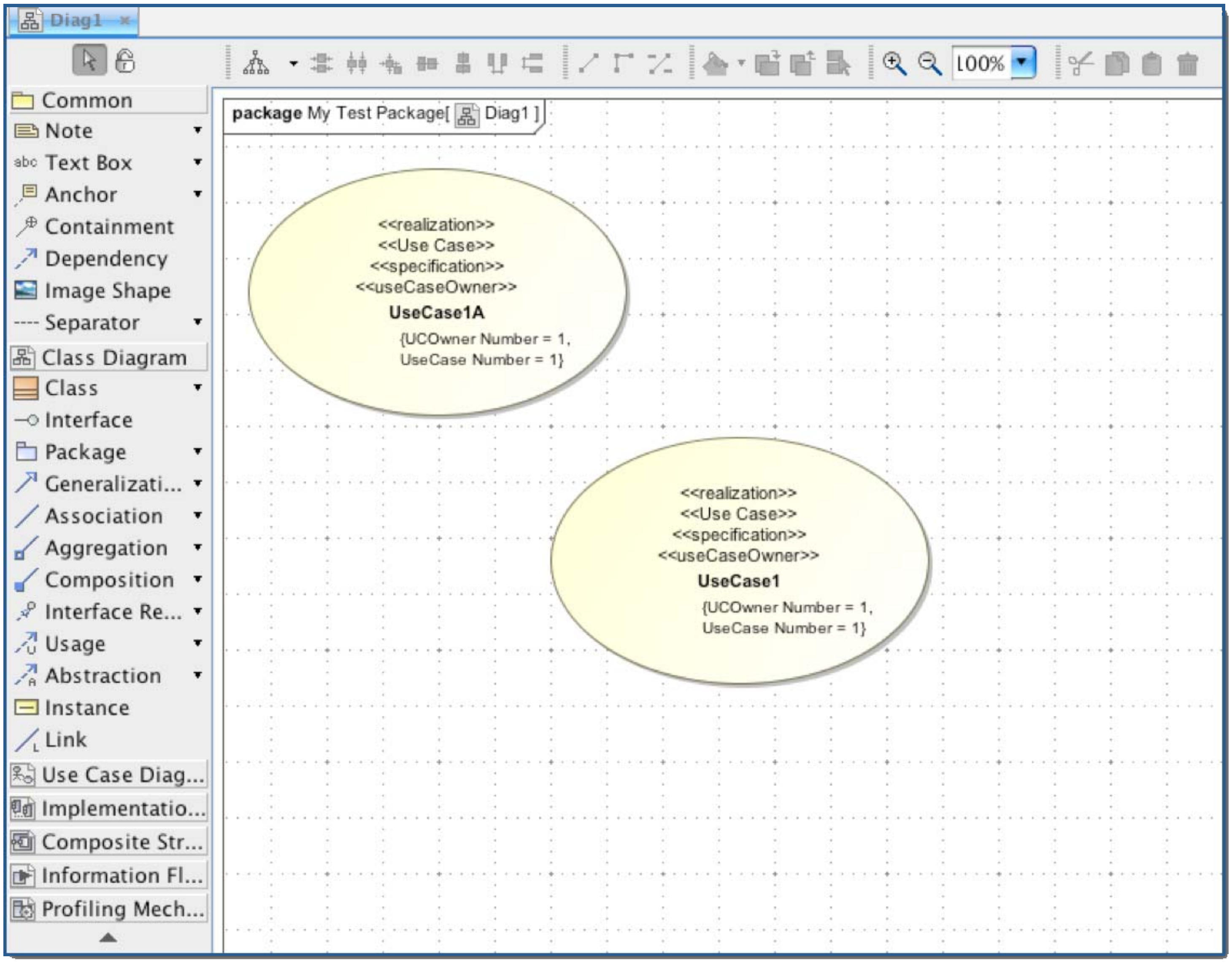
Property Name	Owner	Column Index
Diagram		0
name	UseCase	1
PreCondition	requirementUseCase	3
PostCondition	requirementUseCase	4
Out-standing Issue	requirementUseCase	5
Notes	requirementUseCase	6
Assumption	requirementUseCase	7

Key property: name

After you press Finish, your Containment area should look like this:

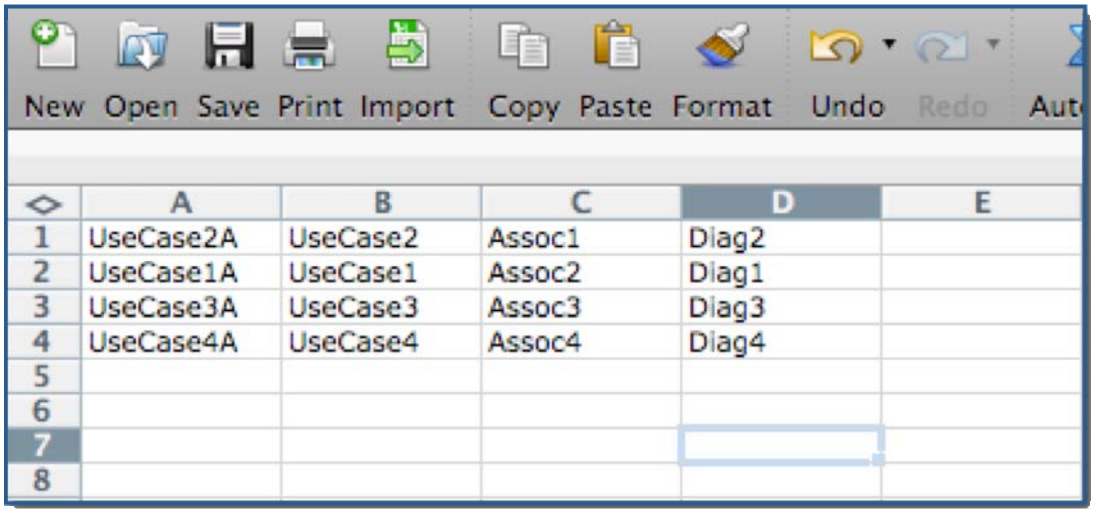

Containment
Data
My Test Package
UseCase1
UseCase1A
UseCase2
UseCase2A
UseCase3
UseCase3A
UseCase4
UseCase4A
Diag1
Diag2
Diag3
Diag4
UML Standard Profile [UML_Standard_Profile.

If you open up Diag1, you should see something similar to this:



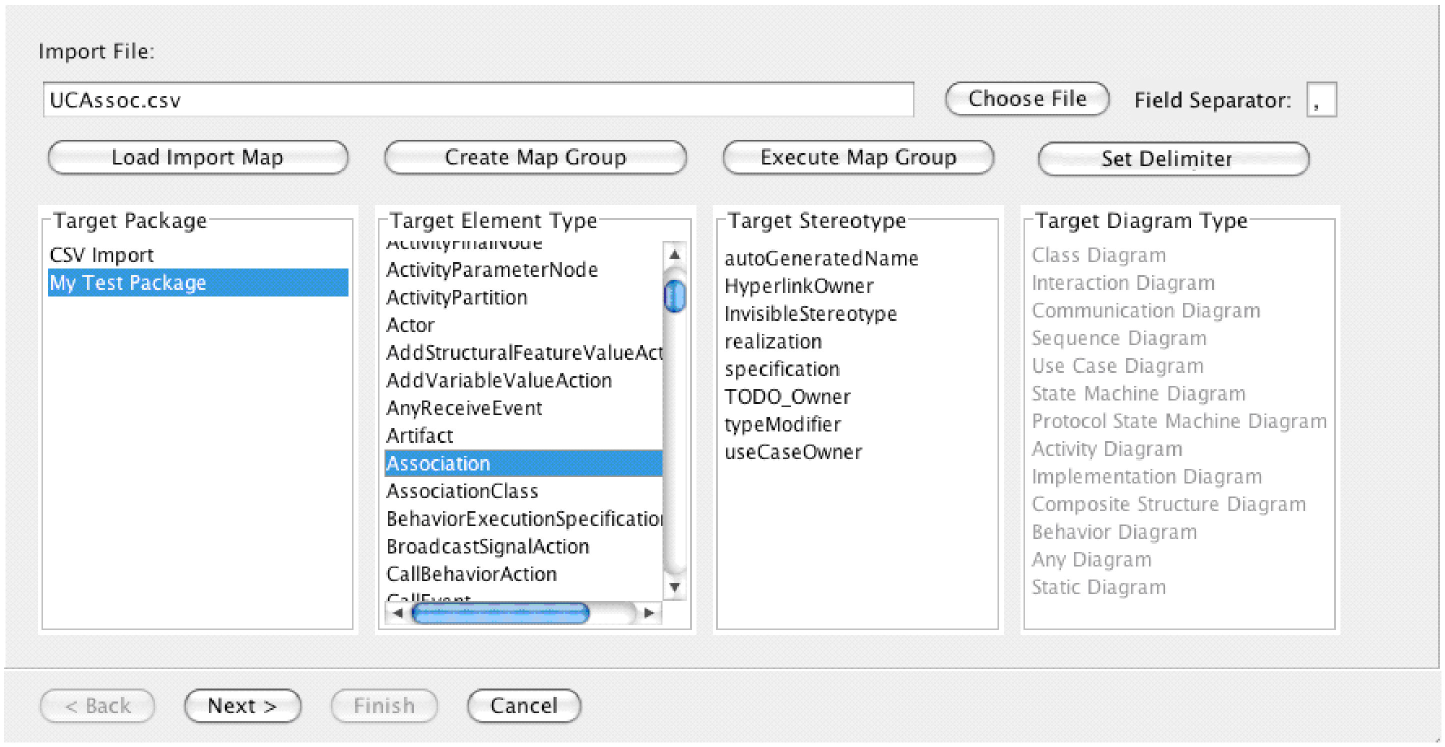
Creating Associations into Diagram

Now, we will add associations between these elements. Start out with spreadsheet data:



	A	B	C	D	E
1	UseCase2A	UseCase2	Assoc1	Diag2	
2	UseCase1A	UseCase1	Assoc2	Diag1	
3	UseCase3A	UseCase3	Assoc3	Diag3	
4	UseCase4A	UseCase4	Assoc4	Diag4	
5					
6					
7					
8					

Then, bring up the import plug-in


Import File: UCAssoc.csv

Field Separator: ,

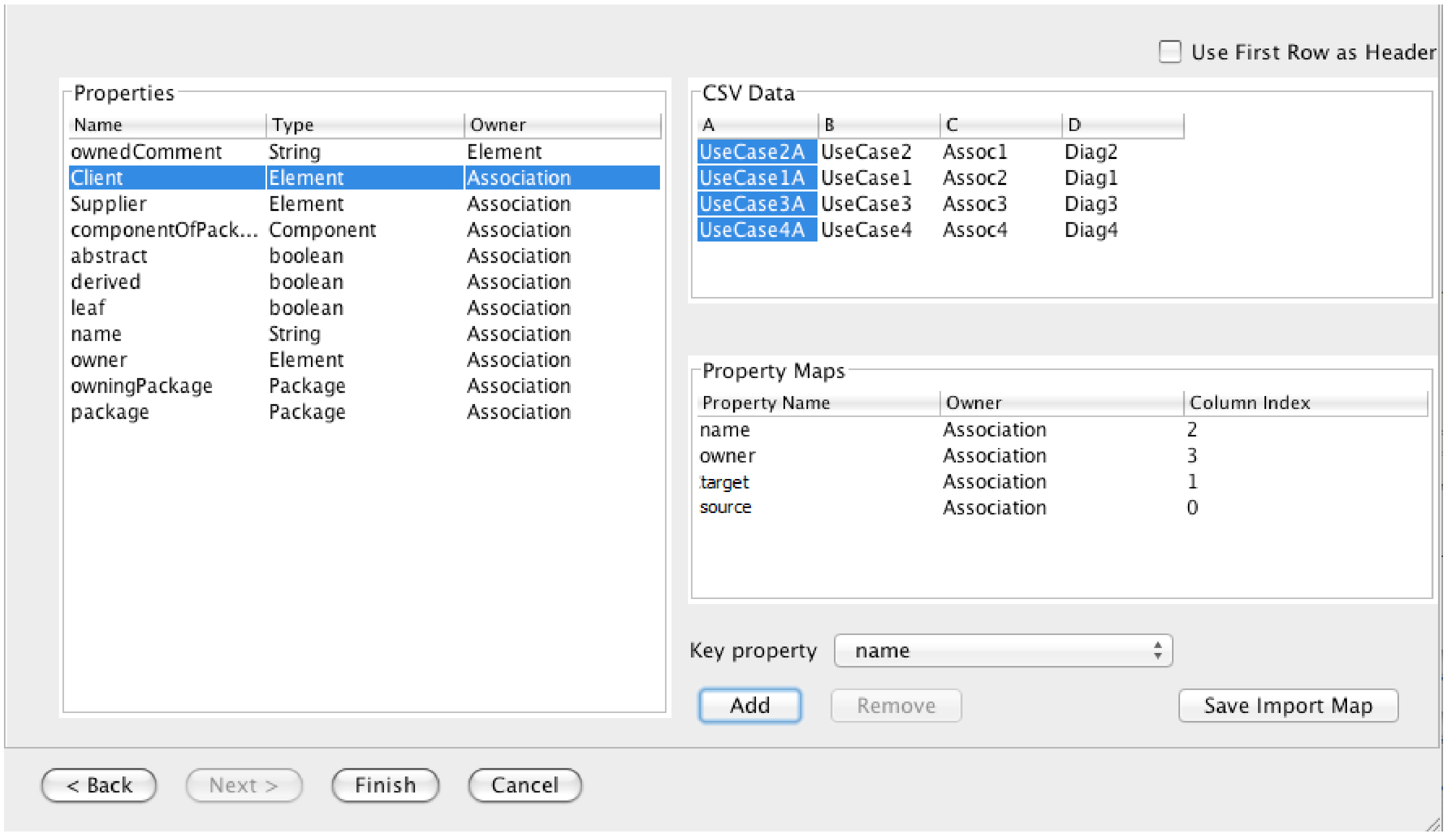
Target Package
- CSV Import
- My Test Package

Target Element Type
- ActivityFinalNode
- ActivityParameterNode
- ActivityPartition
- Actor
- AddStructuralFeatureValueAct
- AddVariableValueAction
- AnyReceiveEvent
- Artifact
- Association
- AssociationClass
- BehaviorExecutionSpecificatio
- BroadcastSignalAction
- CallBehaviorAction
- CallEvent

Target Stereotype
- autoGeneratedName
- HyperlinkOwner
- InvisibleStereotype
- realization
- specification
- TODO_Owner
- typeModifier
- useCaseOwner

Target Diagram Type
- Class Diagram
- Interaction Diagram
- Communication Diagram
- Sequence Diagram
- Use Case Diagram
- State Machine Diagram
- Protocol State Machine Diagram
- Activity Diagram
- Implementation Diagram
- Composite Structure Diagram
- Behavior Diagram
- Any Diagram
- Static Diagram

Press Next to move to the next screen.


Use First Row as Header

Properties

Name	Type	Owner
ownedComment	String	Element
Client	Element	Association
Supplier	Element	Association
componentOfPack...	Component	Association
abstract	boolean	Association
derived	boolean	Association
leaf	boolean	Association
name	String	Association
owner	Element	Association
owningPackage	Package	Association
package	Package	Association

CSV Data

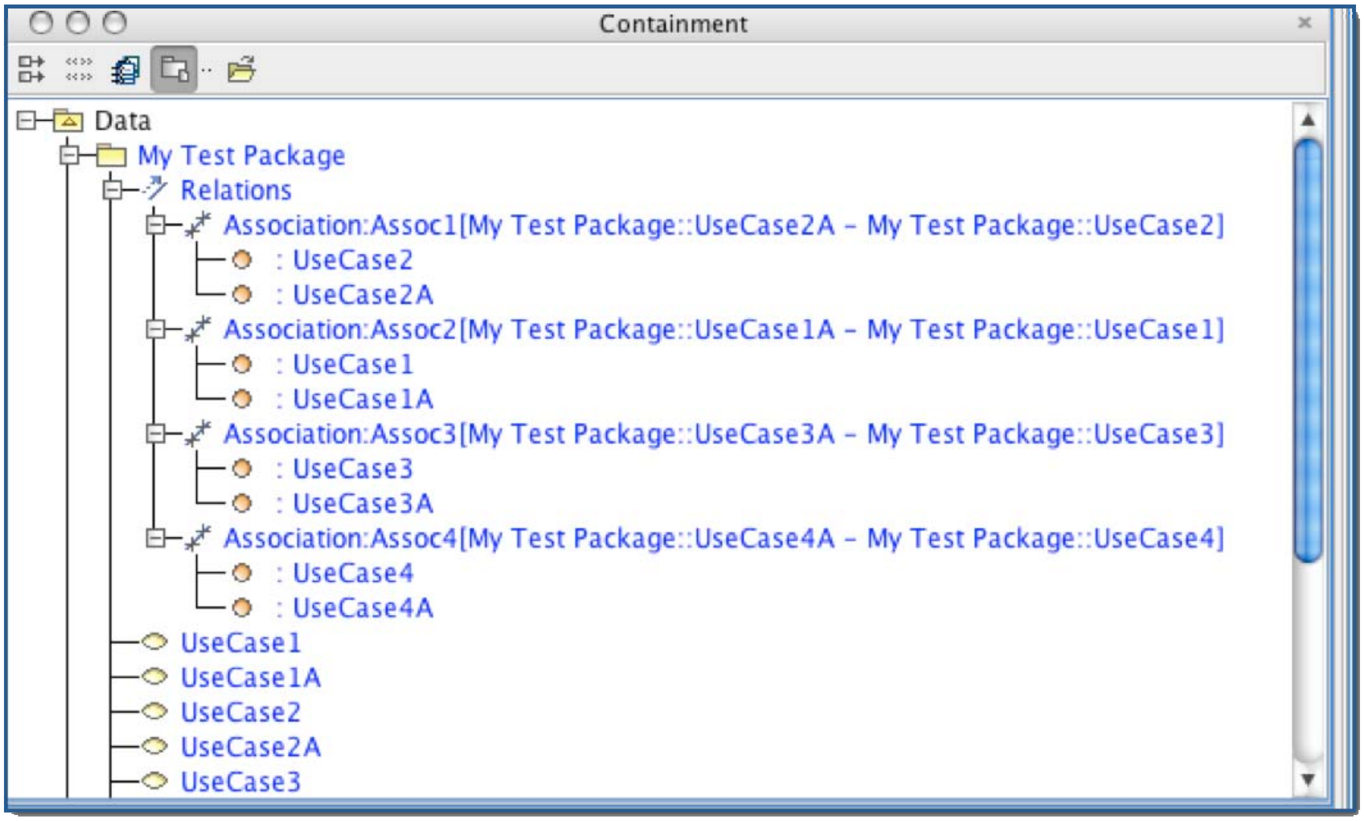
A	B	C	D
UseCase2A	UseCase2	Assoc1	Diag2
UseCase1A	UseCase1	Assoc2	Diag1
UseCase3A	UseCase3	Assoc3	Diag3
UseCase4A	UseCase4	Assoc4	Diag4

Property Maps

Property Name	Owner	Column Index
name	Association	2
owner	Association	3
target	Association	1
source	Association	0

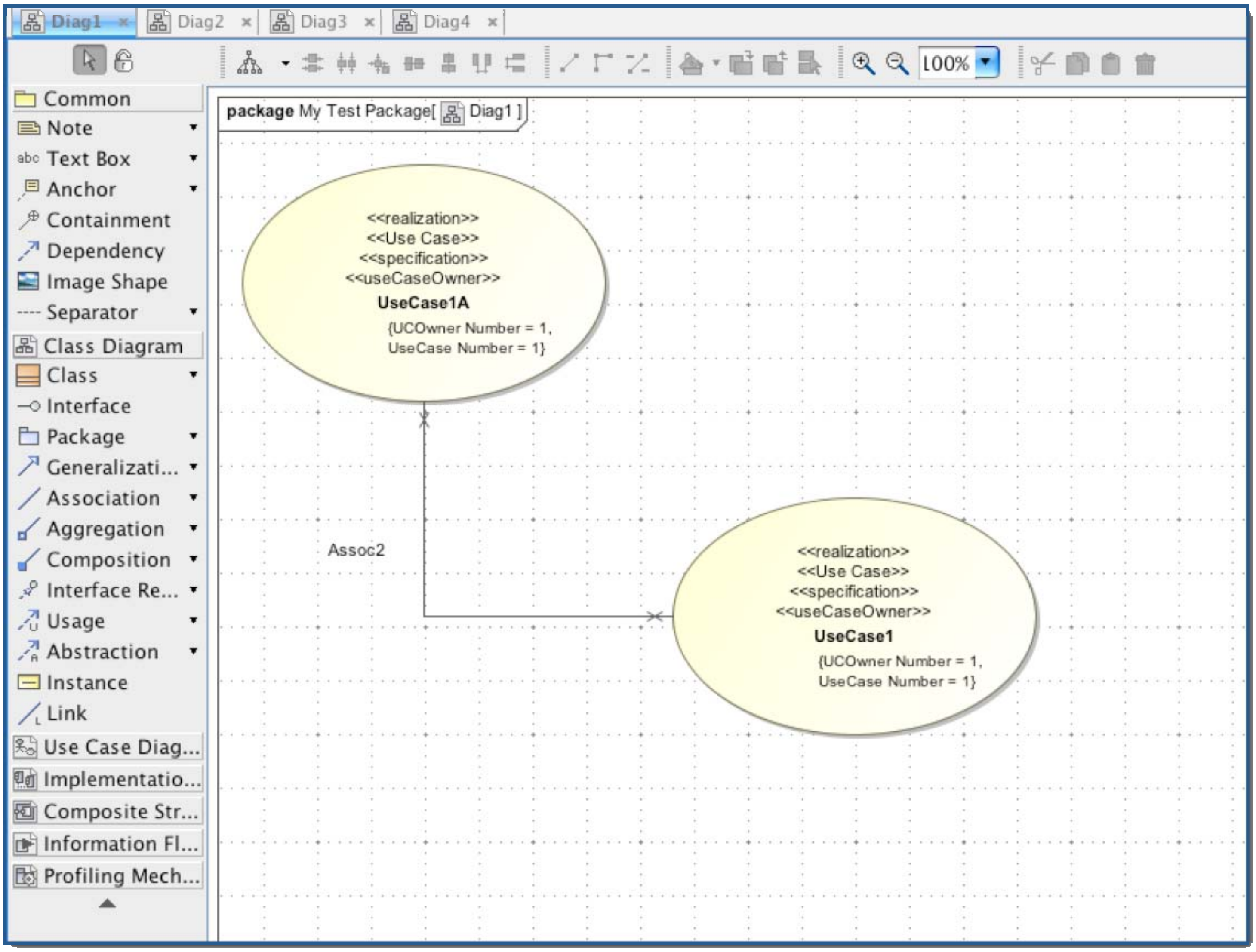
Key property: name

Again, your containment area reflects the new additions to your model:



```
Data
  My Test Package
    Relations
      Association:Assoc1[My Test Package::UseCase2A – My Test Package::UseCase2]
        : UseCase2
        : UseCase2A
      Association:Assoc2[My Test Package::UseCase1A – My Test Package::UseCase1]
        : UseCase1
        : UseCase1A
      Association:Assoc3[My Test Package::UseCase3A – My Test Package::UseCase3]
        : UseCase3
        : UseCase3A
      Association:Assoc4[My Test Package::UseCase4A – My Test Package::UseCase4]
        : UseCase4
        : UseCase4A
    UseCase1
    UseCase1A
    UseCase2
    UseCase2A
    UseCase3
```

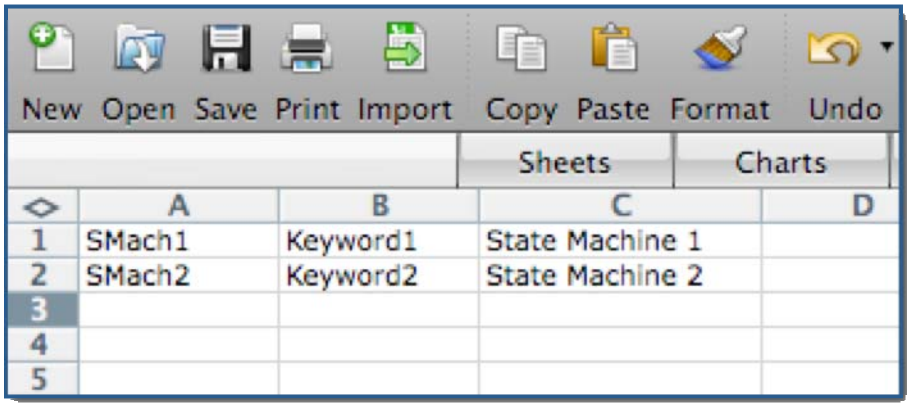
Now, see how your diagram has changed:



Creating and Populating State Diagrams

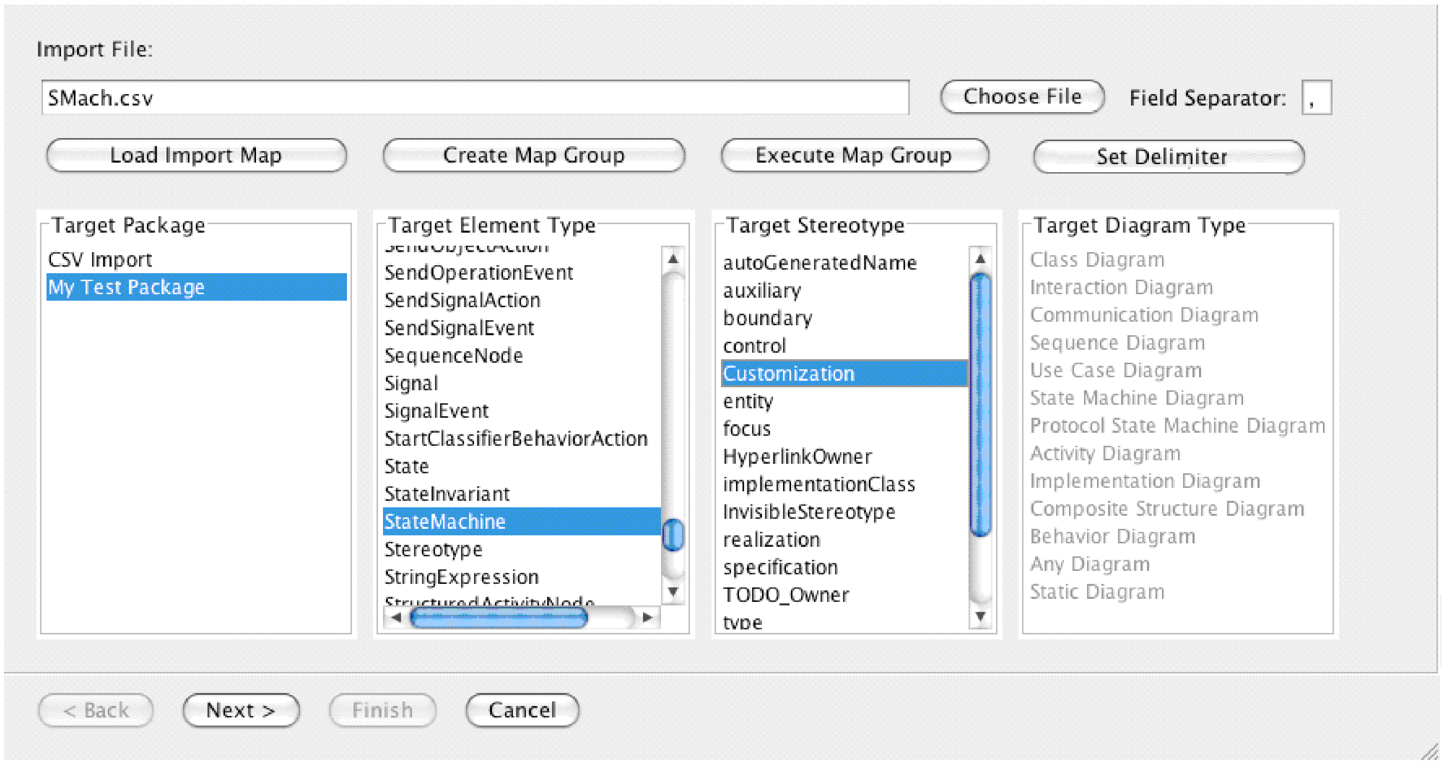
State diagrams are unusual in that they are not owned by the package. Therefore, there are some extra steps you need to perform to import these diagrams. This is also the case with Activity and Sequence diagrams.

Step 1: Create State Machine Elements



	A	B	C	D
1	SMach1	Keyword1	State Machine 1	
2	SMach2	Keyword2	State Machine 2	
3				
4				
5				

Bring up the plug-in.



Import File: SMach.csv

Field Separator: ,

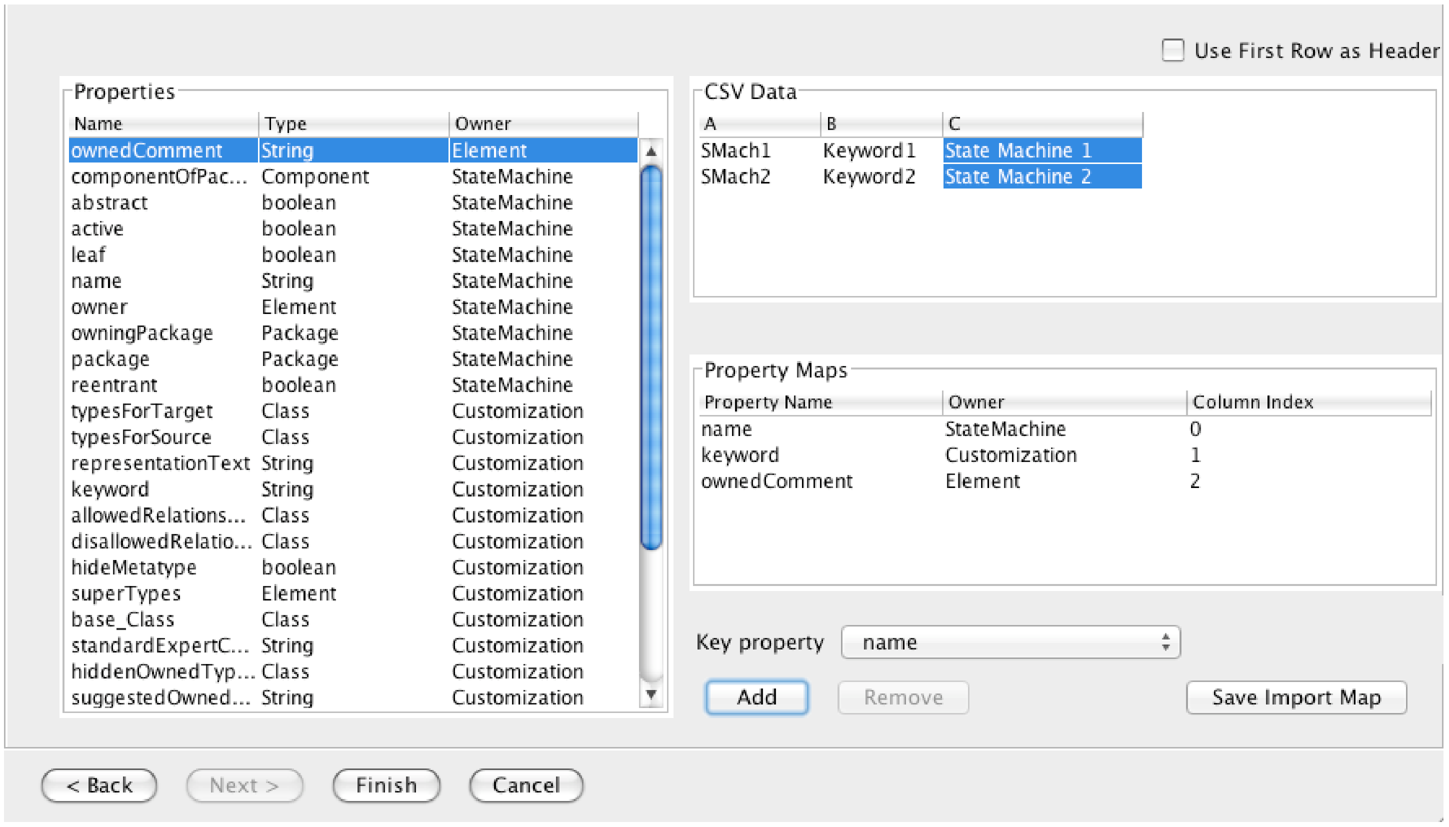
Target Package: CSV Import, My Test Package (selected)

Target Element Type: SendObjectAction, SendOperationEvent, SendSignalAction, SendSignalEvent, SequenceNode, Signal, SignalEvent, StartClassifierBehaviorAction, State, StateInvariant, StateMachine (selected), Stereotype, StringExpression, StructuredActivityNode

Target Stereotype: autoGeneratedName, auxiliary, boundary, control, Customization (selected), entity, focus, HyperlinkOwner, implementationClass, InvisibleStereotype, realization, specification, TODO_Owner, type

Target Diagram Type: Class Diagram, Interaction Diagram, Communication Diagram, Sequence Diagram, Use Case Diagram, State Machine Diagram, Protocol State Machine Diagram, Activity Diagram, Implementation Diagram, Composite Structure Diagram, Behavior Diagram, Any Diagram, Static Diagram

Press Next to move to the next screen.



Properties

Name	Type	Owner
ownedComment	String	Element
componentOfPac...	Component	StateMachine
abstract	boolean	StateMachine
active	boolean	StateMachine
leaf	boolean	StateMachine
name	String	StateMachine
owner	Element	StateMachine
owningPackage	Package	StateMachine
package	Package	StateMachine
reentrant	boolean	StateMachine
typesForTarget	Class	Customization
typesForSource	Class	Customization
representationText	String	Customization
keyword	String	Customization
allowedRelations...	Class	Customization
disallowedRelatio...	Class	Customization
hideMetatype	boolean	Customization
superTypes	Element	Customization
base_Class	Class	Customization
standardExpertC...	String	Customization
hiddenOwnedTyp...	Class	Customization
suggestedOwned...	String	Customization

CSV Data

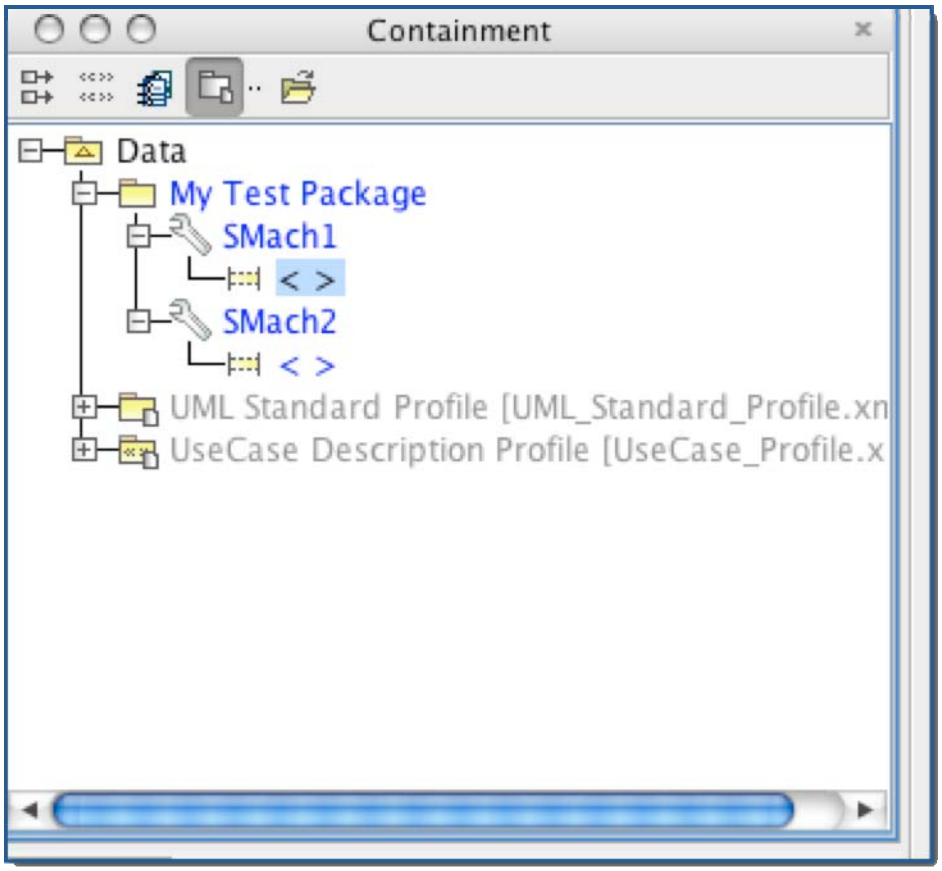
A	B	C
SMach1	Keyword1	State Machine 1
SMach2	Keyword2	State Machine 2

Property Maps

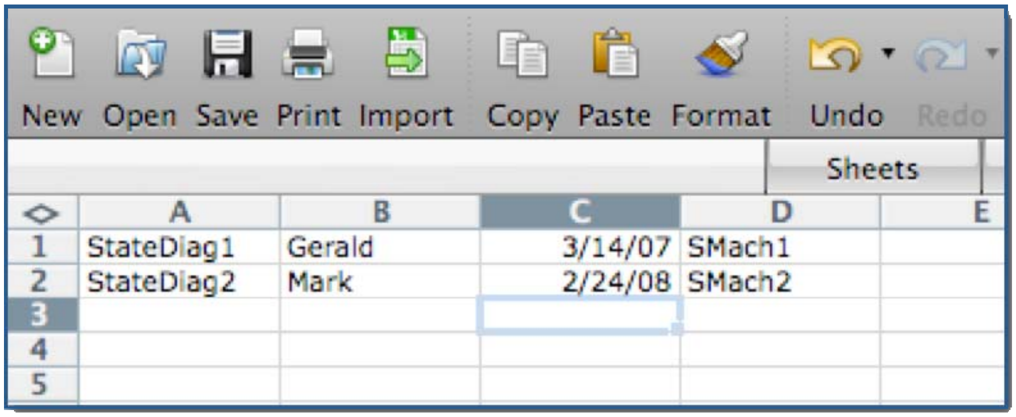
Property Name	Owner	Column Index
name	StateMachine	0
keyword	Customization	1
ownedComment	Element	2

Key property: name

Press Finish and your containment area should look like this:

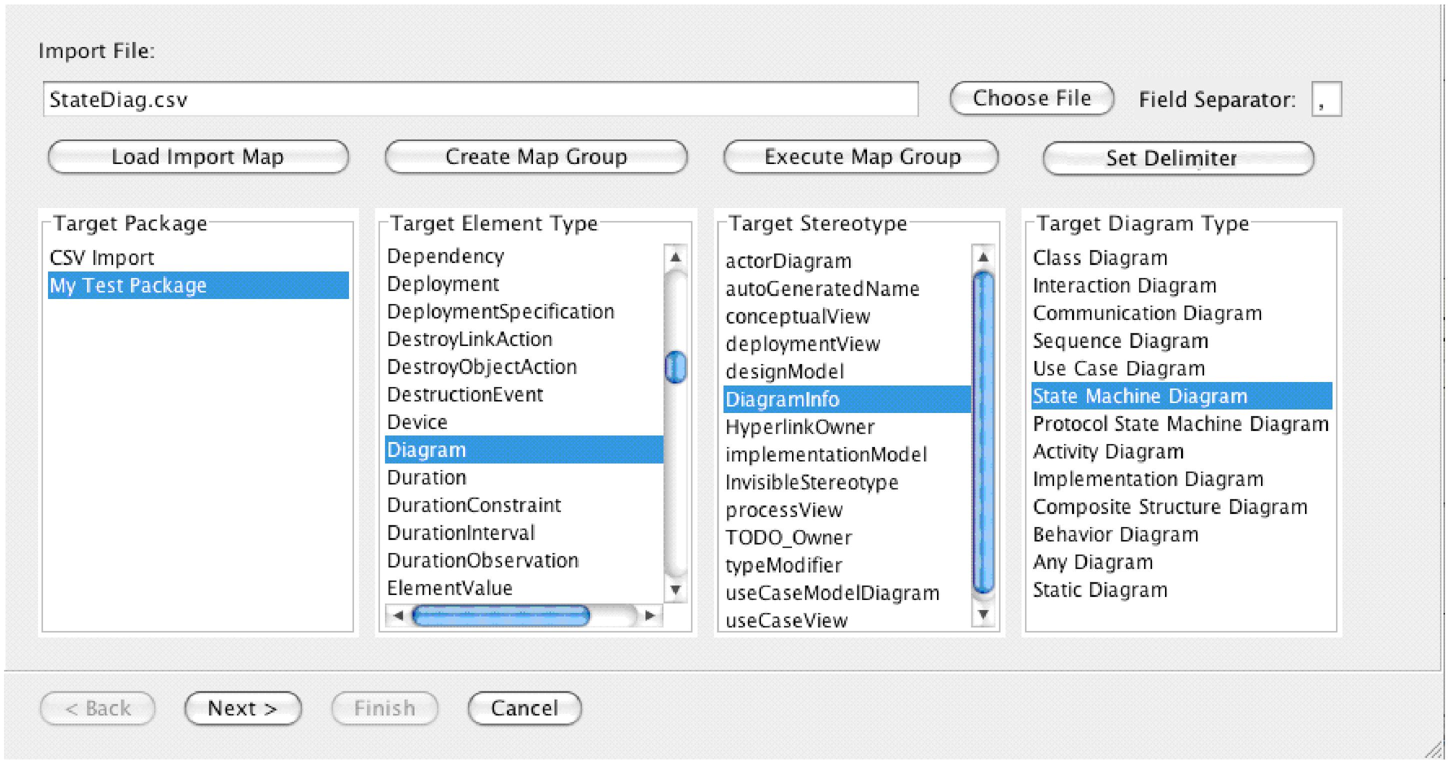

Containment
Data
My Test Package
SMach1
< >
SMach2
< >
UML Standard Profile [UML_Standard_Profile.xn
UseCase Description Profile [UseCase_Profile.x

Step 2: Create State Machine Diagrams



	A	B	C	D	E
1	StateDiag1	Gerald	3/14/07	SMach1	
2	StateDiag2	Mark	2/24/08	SMach2	
3					
4					
5					

Step 3: Bring up the plug-in


Import File: StateDiag.csv

Field Separator: ,

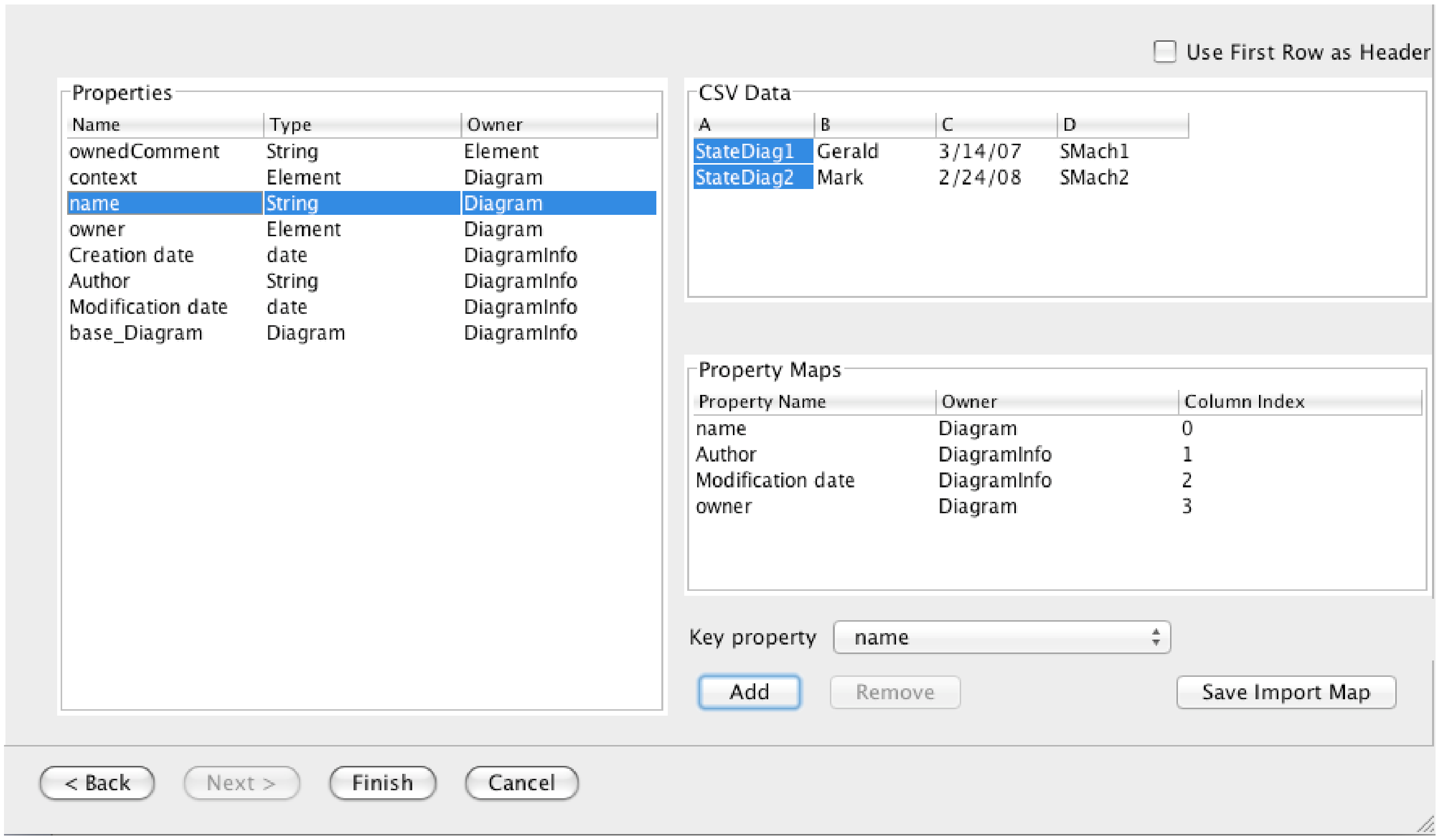
Target Package: CSV Import, My Test Package (selected)

Target Element Type: Dependency, Deployment, DeploymentSpecification, DestroyLinkAction, DestroyObjectAction, DestructionEvent, Device, Diagram (selected), Duration, DurationConstraint, DurationInterval, DurationObservation, ElementValue

Target Stereotype: actorDiagram, autoGeneratedName, conceptualView, deploymentView, designModel, DiagramInfo (selected), HyperlinkOwner, implementationModel, InvisibleStereotype, processView, TODO_Owner, typeModifier, useCaseModelDiagram, useCaseView

Target Diagram Type: Class Diagram, Interaction Diagram, Communication Diagram, Sequence Diagram, Use Case Diagram, State Machine Diagram (selected), Protocol State Machine Diagram, Activity Diagram, Implementation Diagram, Composite Structure Diagram, Behavior Diagram, Any Diagram, Static Diagram

Step 4: Press Next to move to the next screen.


Use First Row as Header

Properties

Name	Type	Owner
ownedComment	String	Element
context	Element	Diagram
name	String	Diagram
owner	Element	Diagram
Creation date	date	DiagramInfo
Author	String	DiagramInfo
Modification date	date	DiagramInfo
base_Diagram	Diagram	DiagramInfo

CSV Data

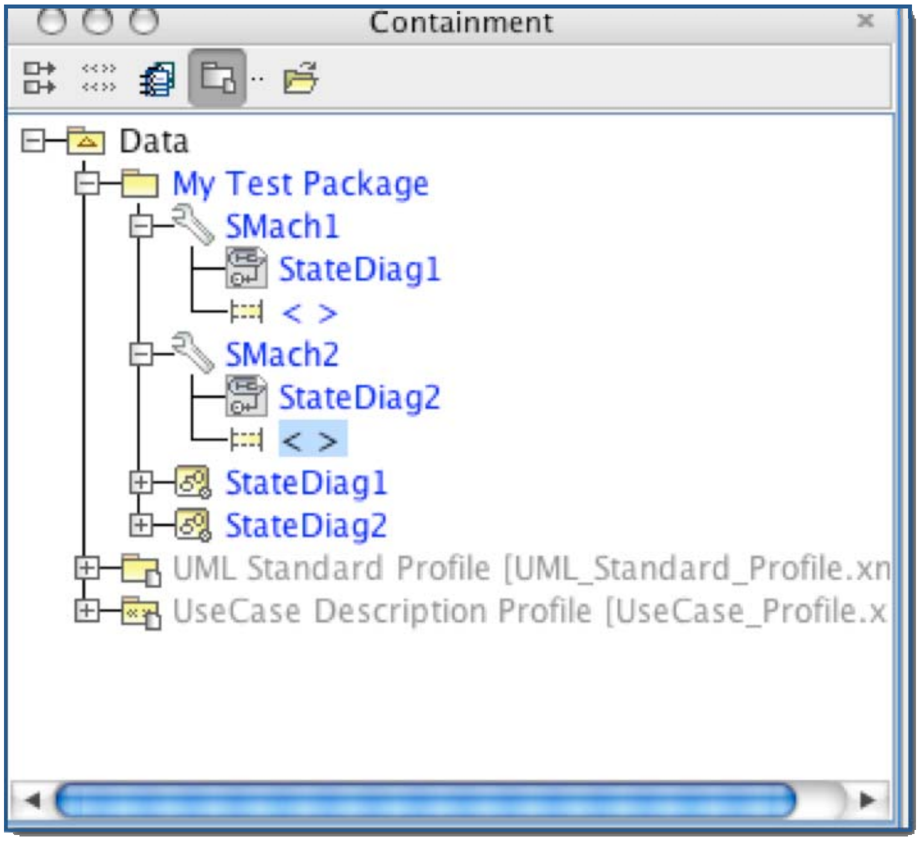
A	B	C	D
StateDiag1	Gerald	3/14/07	SMach1
StateDiag2	Mark	2/24/08	SMach2

Property Maps

Property Name	Owner	Column Index
name	Diagram	0
Author	DiagramInfo	1
Modification date	DiagramInfo	2
owner	Diagram	3

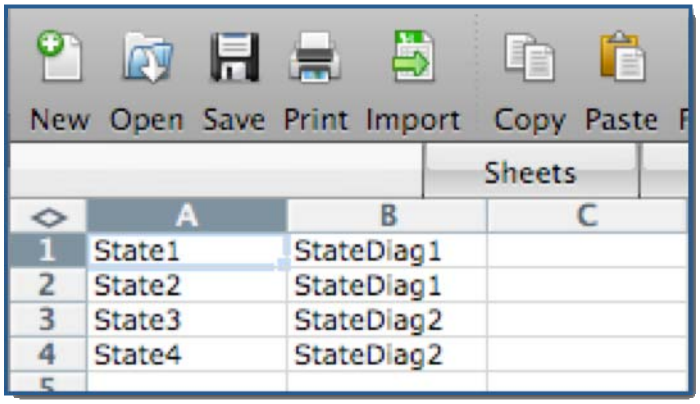
Key property: name

Step 5: Press Finish and your containment area should look like this:


Containment
Data
My Test Package
SMach1
StateDiag1
< >
SMach2
StateDiag2
< >
StateDiag1
StateDiag2
UML Standard Profile [UML_Standard_Profile.xn
UseCase Description Profile [UseCase_Profile.x

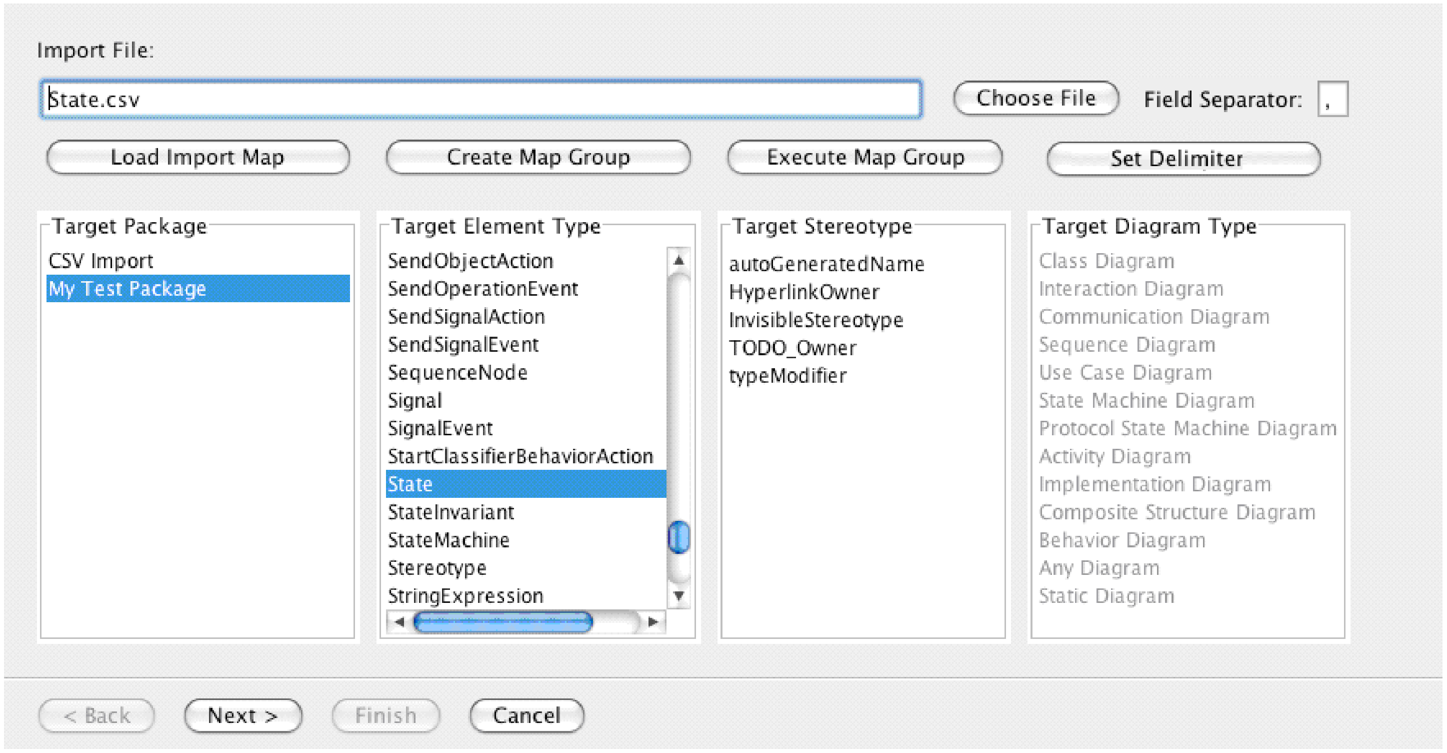
Step 6: Manually delete any superfluous State Diagrams. MagicDraw may, at this point, rename your state machine elements to match the diagram name.

Step 7: Create state elements.



	A	B	C
1	State1	StateDiag1	
2	State2	StateDiag1	
3	State3	StateDiag2	
4	State4	StateDiag2	

Step 8. Import them


Import File: State.csv

Field Separator: ,

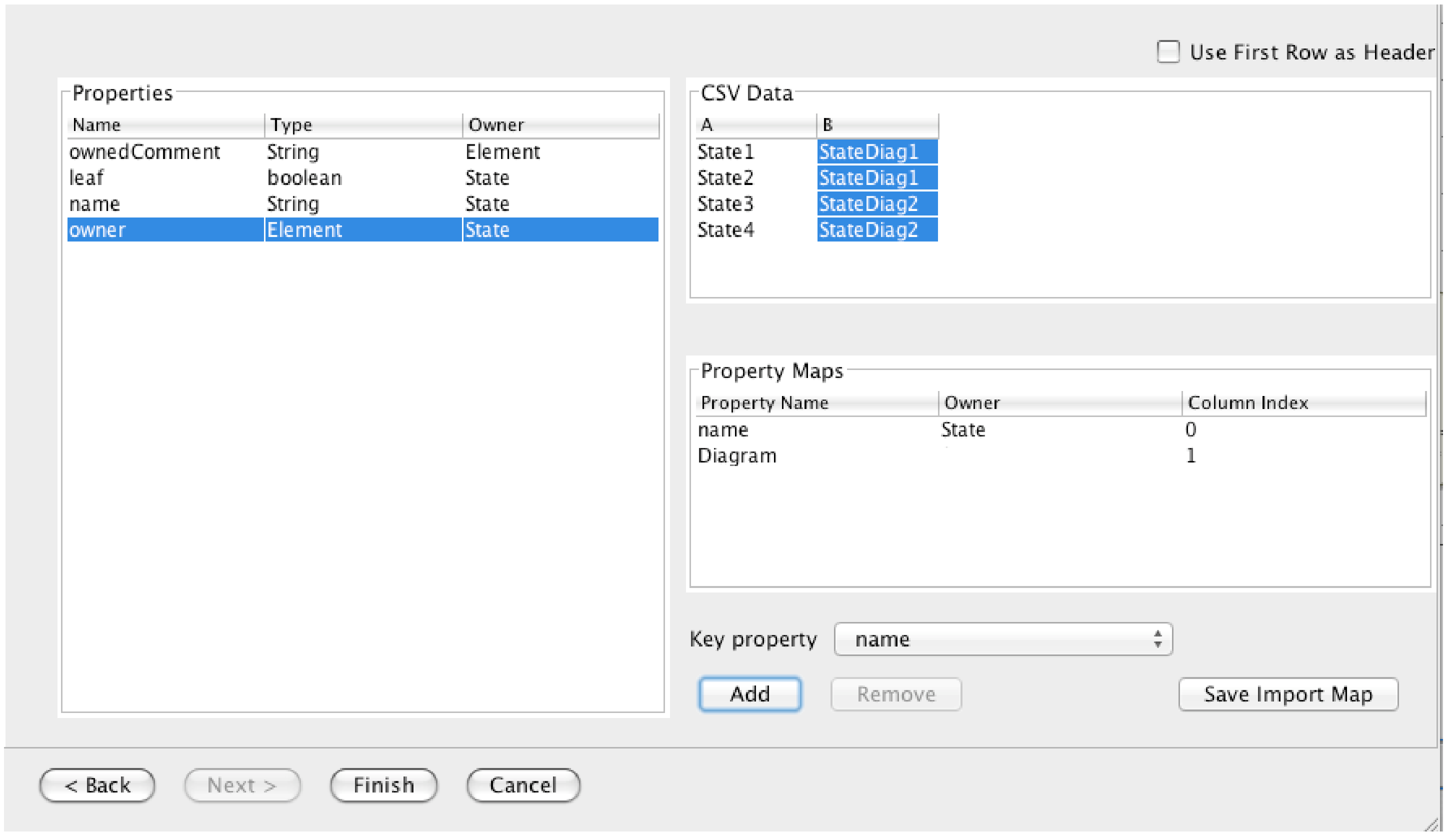
Target Package: CSV Import, My Test Package (selected)

Target Element Type: SendObjectAction, SendOperationEvent, SendSignalAction, SendSignalEvent, SequenceNode, Signal, SignalEvent, StartClassifierBehaviorAction, State (selected), StateInvariant, StateMachine, Stereotype, StringExpression

Target Stereotype: autoGeneratedName, HyperlinkOwner, InvisibleStereotype, TODO_Owner, typeModifier

Target Diagram Type: Class Diagram, Interaction Diagram, Communication Diagram, Sequence Diagram, Use Case Diagram, State Machine Diagram, Protocol State Machine Diagram, Activity Diagram, Implementation Diagram, Composite Structure Diagram, Behavior Diagram, Any Diagram, Static Diagram

Step 9. Press Next to move to the next screen.



Properties

Name	Type	Owner
ownedComment	String	Element
leaf	boolean	State
name	String	State
owner	Element	State

CSV Data

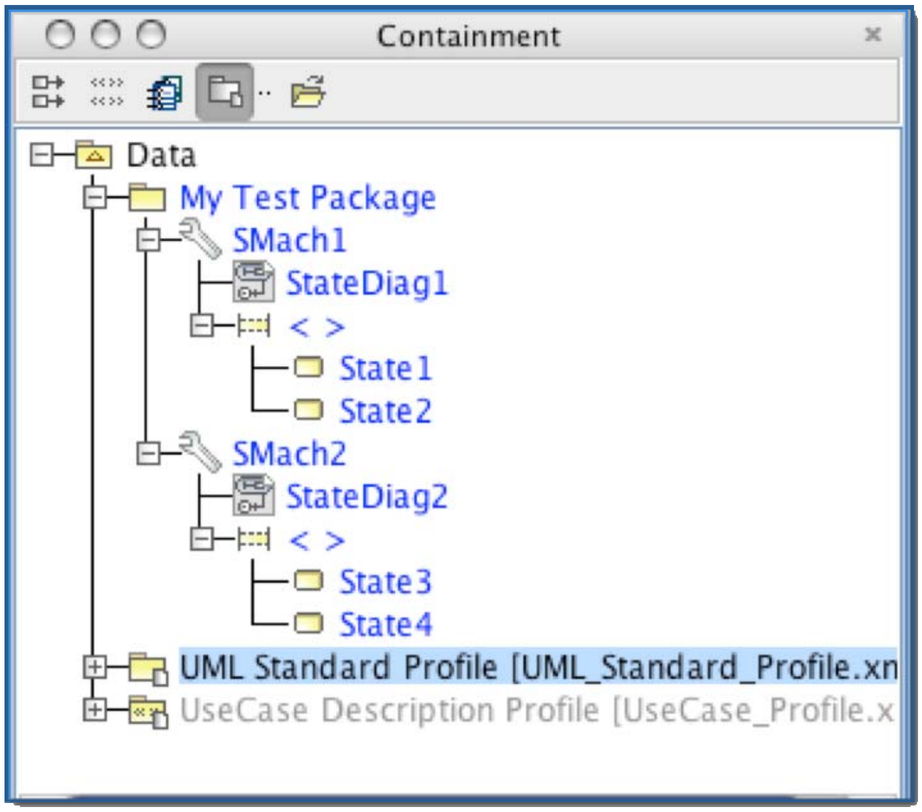
A	B
State1	StateDiag1
State2	StateDiag1
State3	StateDiag2
State4	StateDiag2

Property Maps

Property Name	Owner	Column Index
name	State	0
Diagram		1

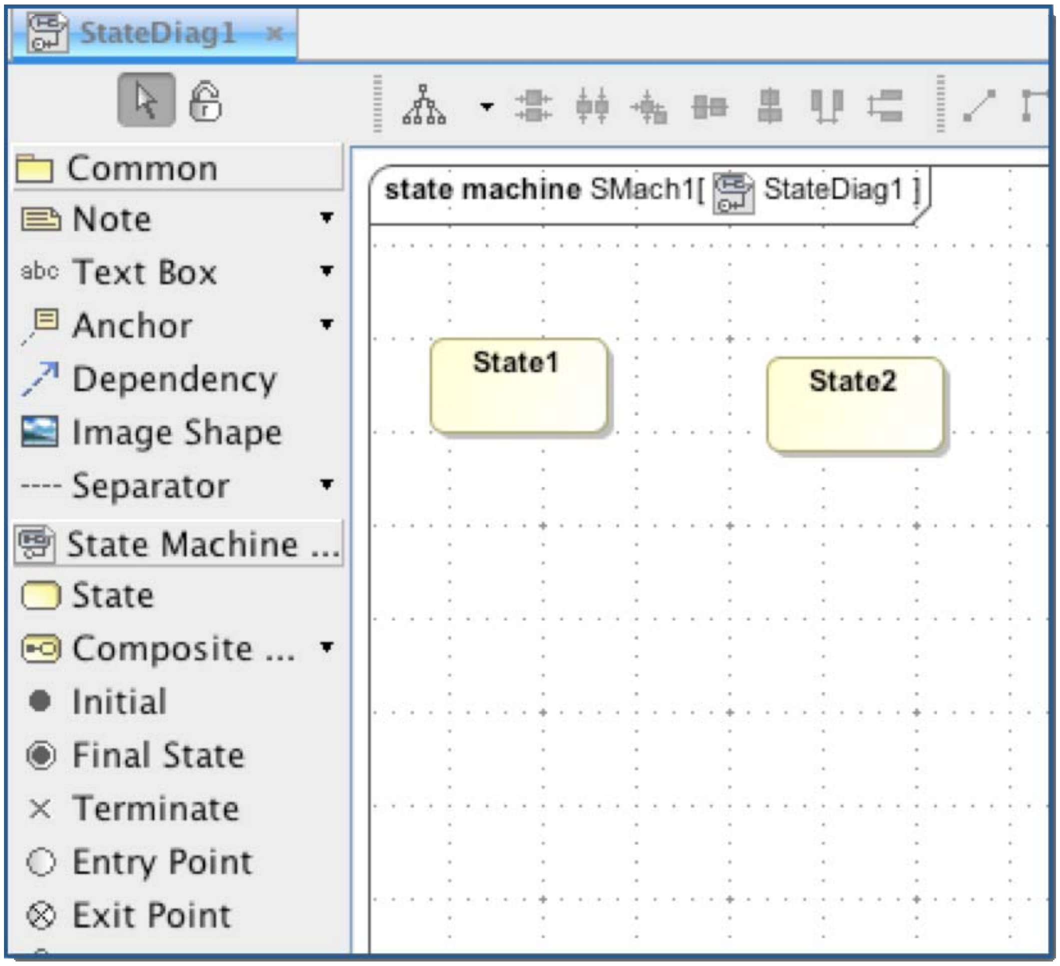
Key property: name

Step 10. Press Finish and review your containment area.

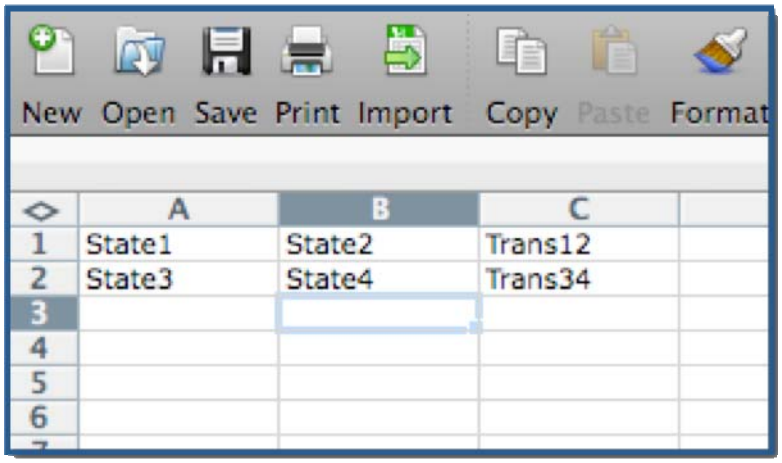


```
Data
  My Test Package
    SMach1
      StateDiag1
      < >
        State1
        State2
    SMach2
      StateDiag2
      < >
        State3
        State4
  UML Standard Profile [UML_Standard_Profile.xn
  UseCase Description Profile [UseCase_Profile.x
```

See them populated on the State Diagrams.

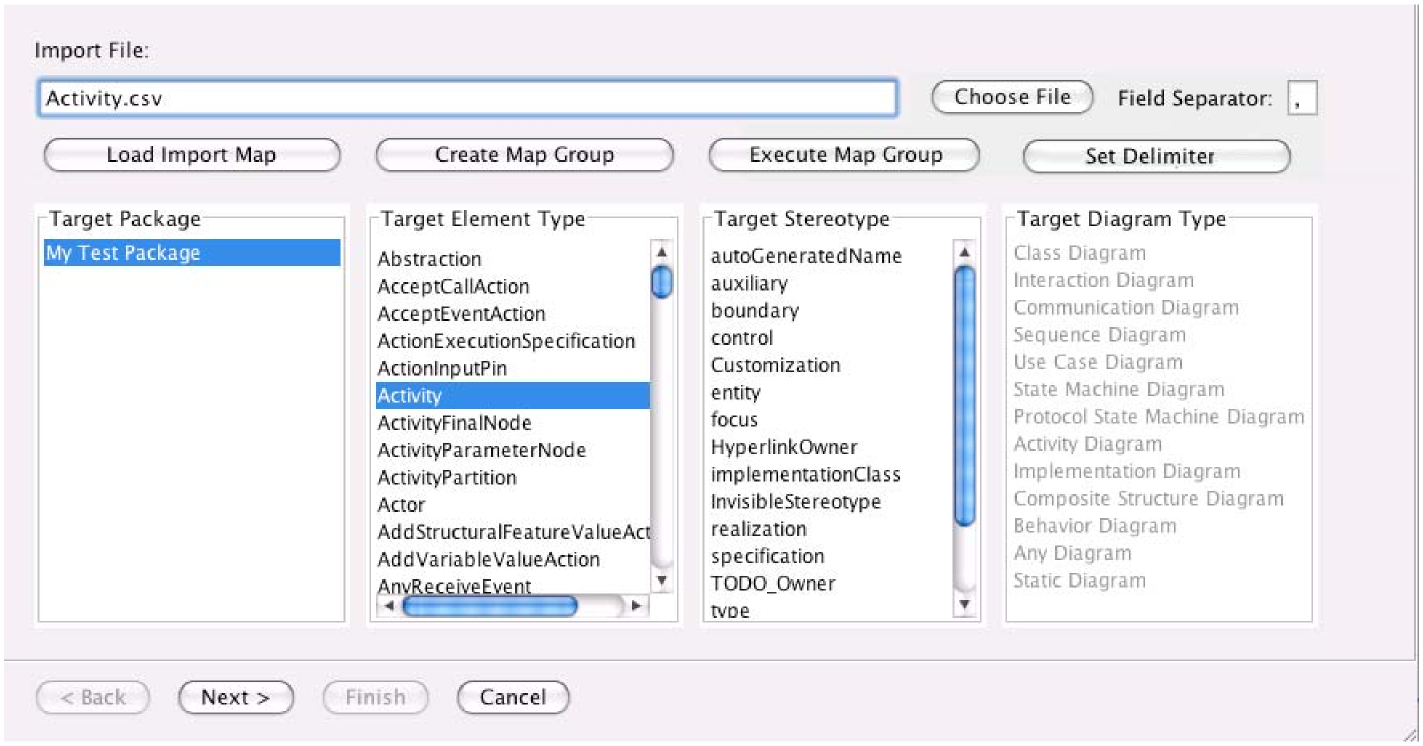


Now, we will import transitions. First, let's create the data:



	A	B	C	
1	State1	State2	Trans12	
2	State3	State4	Trans34	
3				
4				
5				
6				

Now, we'll import it. Bring up the plug-in.



Import File: Activity.csv

Field Separator: ,

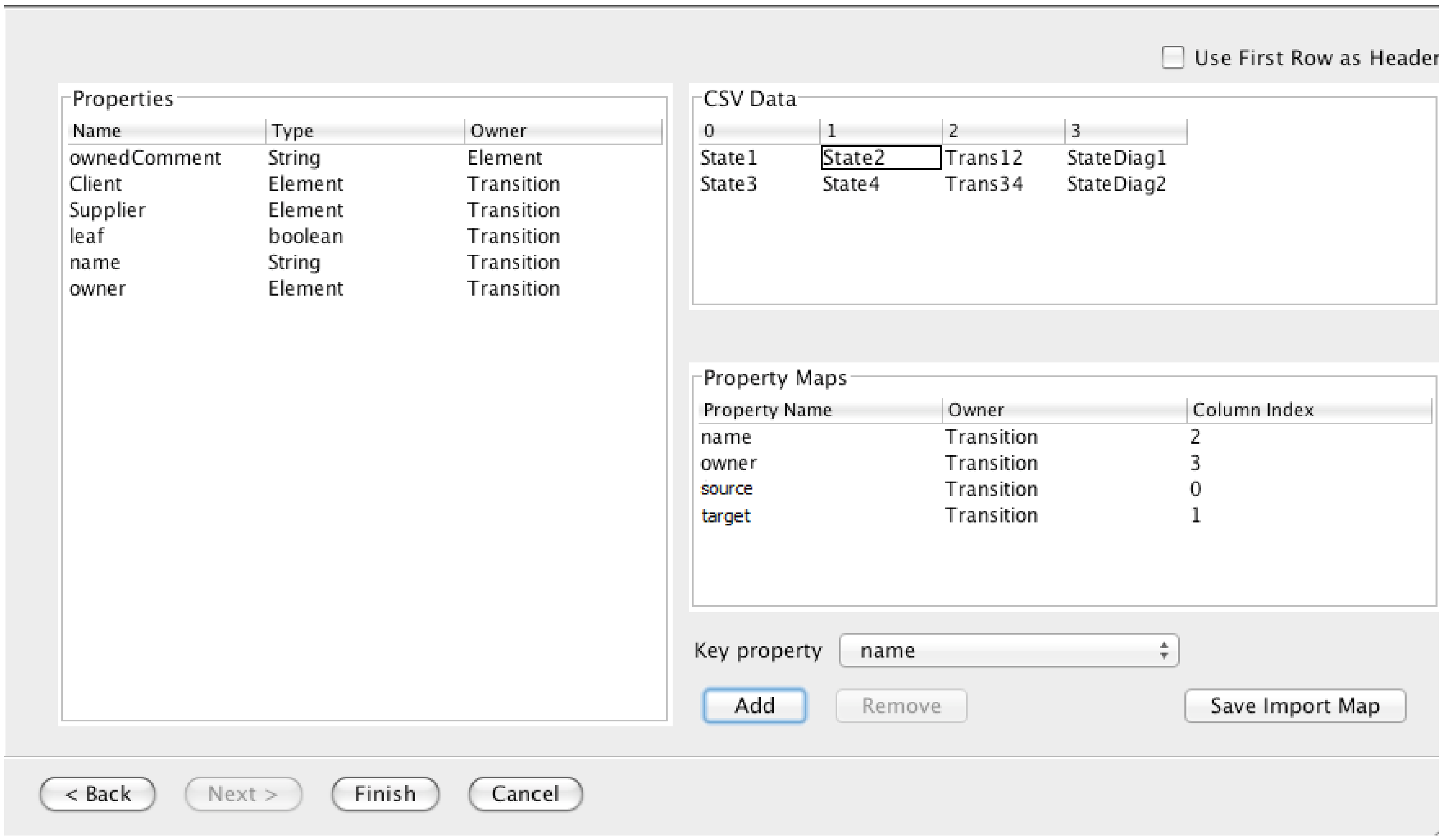
Target Package
- My Test Package

Target Element Type
- Abstraction
- AcceptCallAction
- AcceptEventAction
- ActionExecutionSpecification
- ActionInputPin
- Activity
- ActivityFinalNode
- ActivityParameterNode
- ActivityPartition
- Actor
- AddStructuralFeatureValueAct
- AddVariableValueAction
- AnyReceiveEvent

Target Stereotype
- autoGeneratedName
- auxiliary
- boundary
- control
- Customization
- entity
- focus
- HyperlinkOwner
- implementationClass
- InvisibleStereotype
- realization
- specification
- TODO_Owner
- type

Target Diagram Type
- Class Diagram
- Interaction Diagram
- Communication Diagram
- Sequence Diagram
- Use Case Diagram
- State Machine Diagram
- Protocol State Machine Diagram
- Activity Diagram
- Implementation Diagram
- Composite Structure Diagram
- Behavior Diagram
- Any Diagram
- Static Diagram

Press Next to move to the next screen.



Properties

Name	Type	Owner
ownedComment	String	Element
Client	Element	Transition
Supplier	Element	Transition
leaf	boolean	Transition
name	String	Transition
owner	Element	Transition

CSV Data

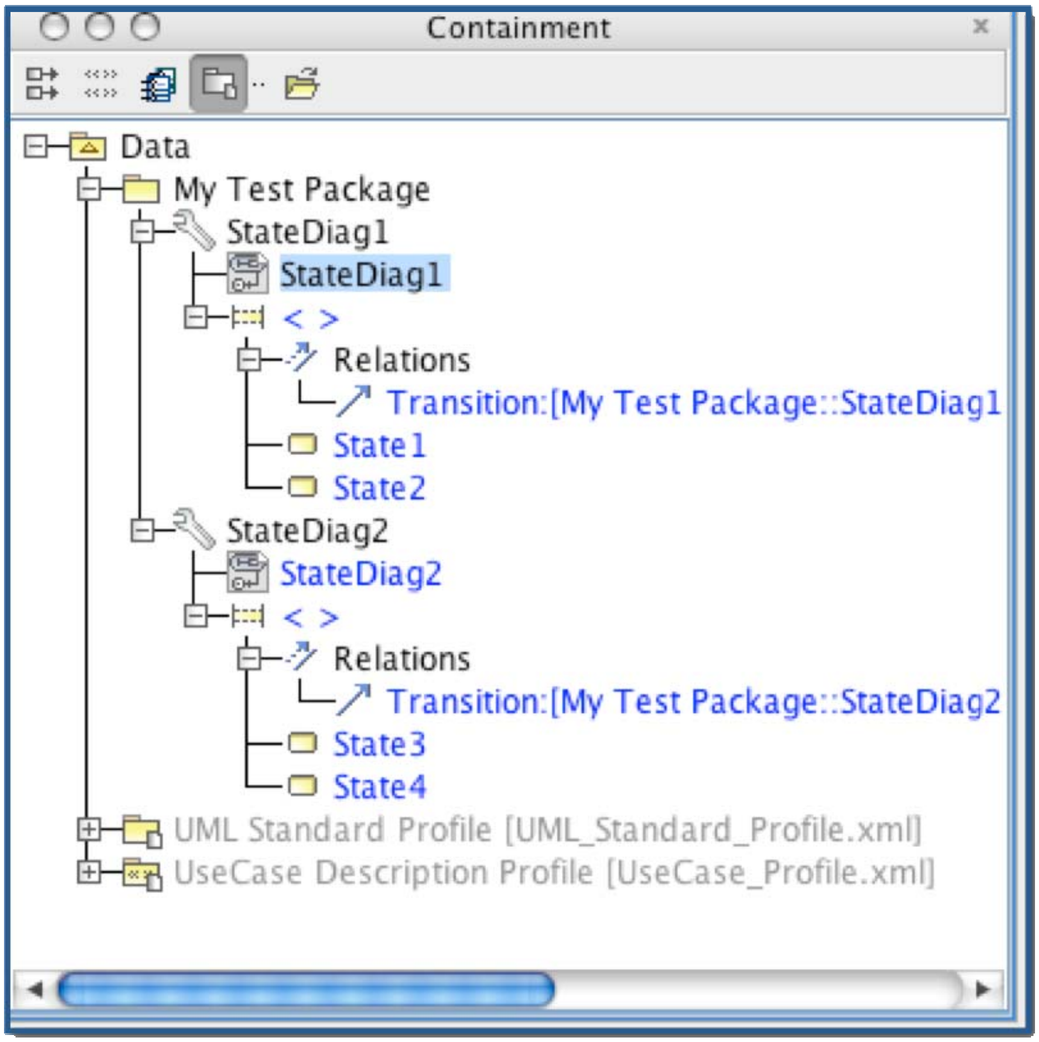
0	1	2	3
State1	State2	Trans12	StateDiag1
State3	State4	Trans34	StateDiag2

Property Maps

Property Name	Owner	Column Index
name	Transition	2
owner	Transition	3
source	Transition	0
target	Transition	1

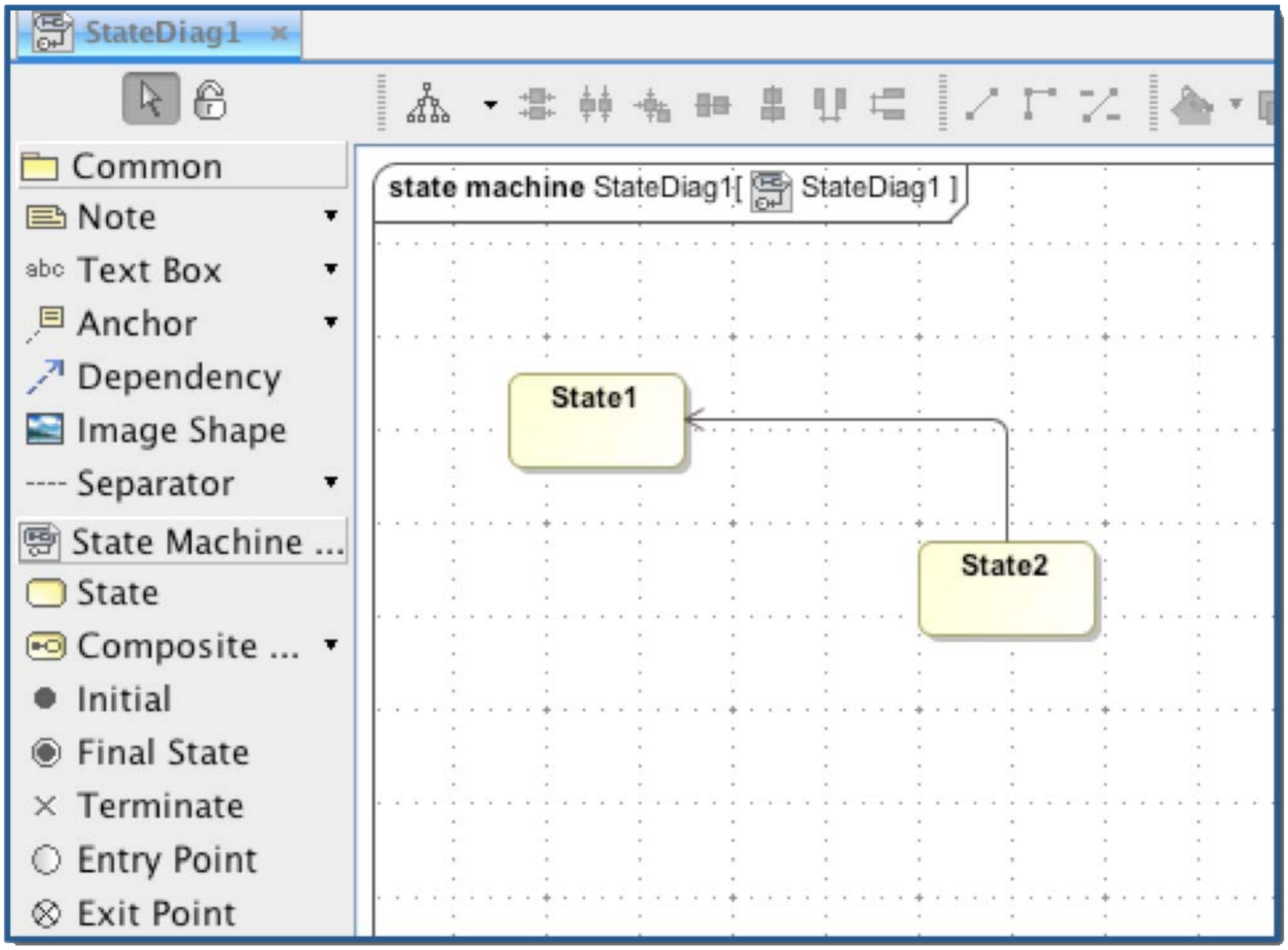
Key property: name

Press Finish and the containment area will look like this:



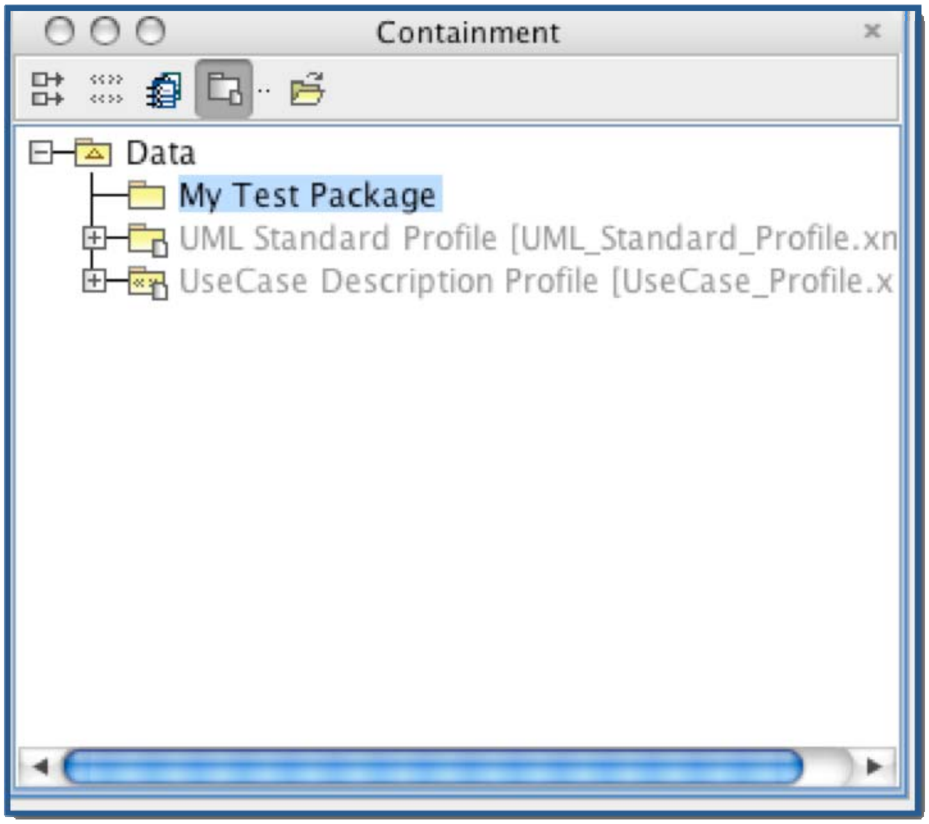
```
Data
  My Test Package
    StateDiag1
      StateDiag1
      < >
        Relations
          Transition:[My Test Package::StateDiag1
        State1
        State2
    StateDiag2
      StateDiag2
      < >
        Relations
          Transition:[My Test Package::StateDiag2
        State3
        State4
  UML Standard Profile [UML_Standard_Profile.xml]
  UseCase Description Profile [UseCase_Profile.xml]
```

A Quick look at StateDiag1 reveals this:

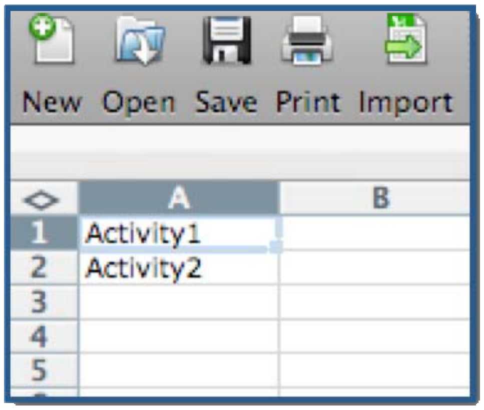


Creating and populating Activity Diagrams

First, we'll start with a clean containment area:

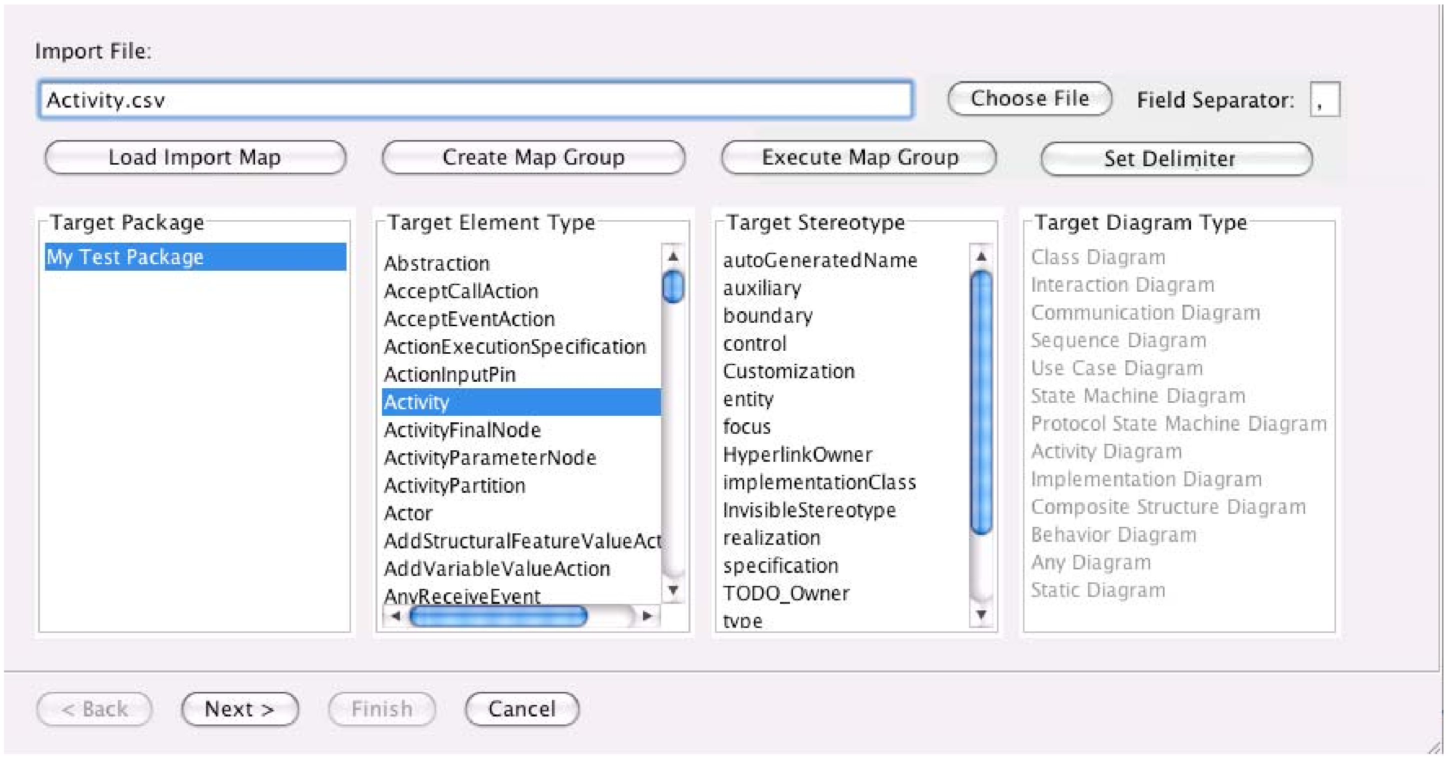

Containment
Data
My Test Package
UML Standard Profile [UML_Standard_Profile.xn
UseCase Description Profile [UseCase_Profile.x

Now, we'll import some activities, since activity diagrams are owned by activity elements:



	A	B
1	Activity1	
2	Activity2	
3		
4		
5		

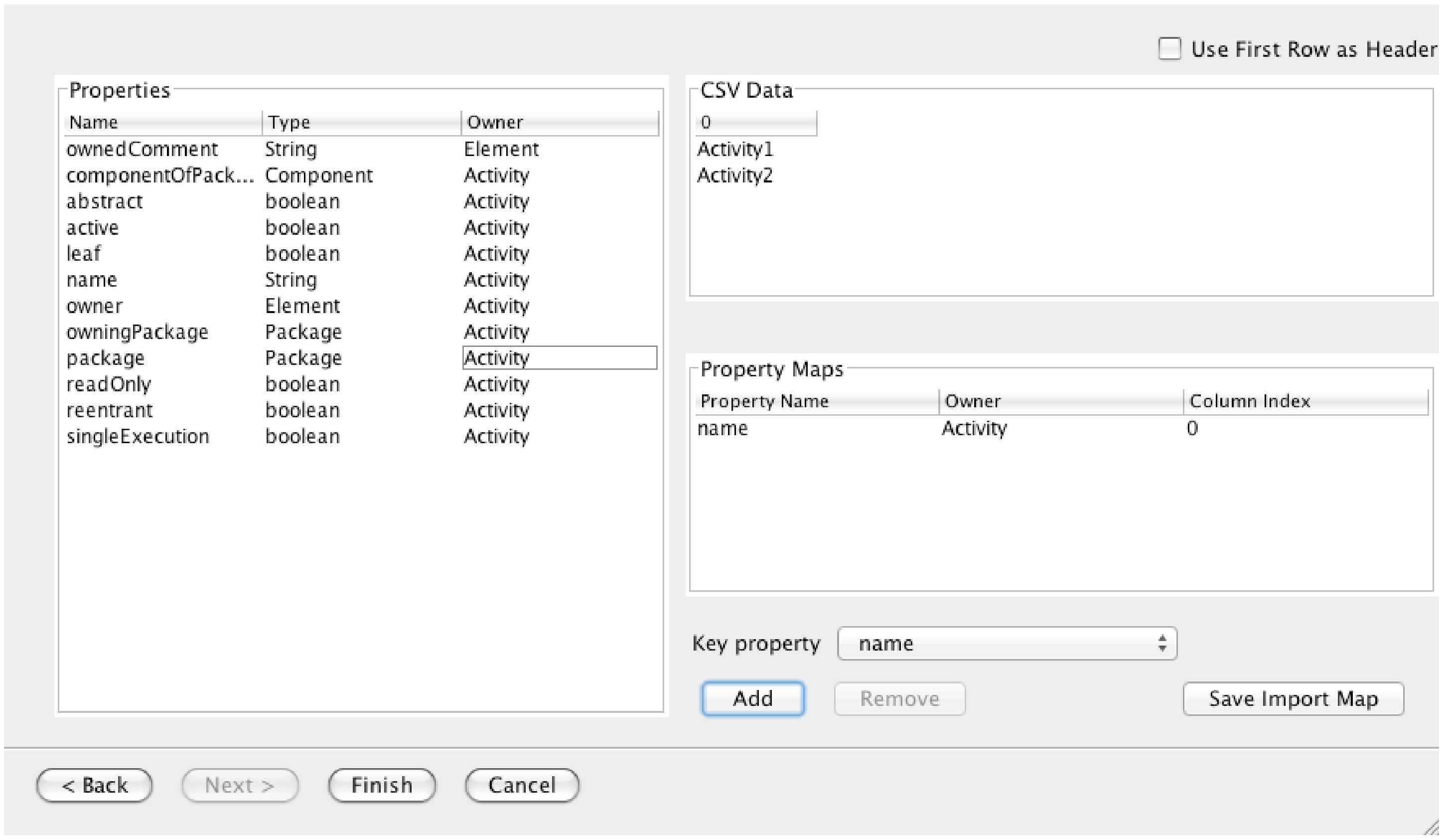
Bring up the plug-in:


Import File: Activity.csv

Field Separator: ,

Target Package	Target Element Type	Target Stereotype	Target Diagram Type
My Test Package	Abstraction	autoGeneratedName	Class Diagram
	AcceptCallAction	auxiliary	Interaction Diagram
	AcceptEventAction	boundary	Communication Diagram
	ActionExecutionSpecification	control	Sequence Diagram
	ActionInputPin	Customization	Use Case Diagram
	Activity	entity	State Machine Diagram
	ActivityFinalNode	focus	Protocol State Machine Diagram
	ActivityParameterNode	HyperlinkOwner	Activity Diagram
	ActivityPartition	implementationClass	Implementation Diagram
	Actor	InvisibleStereotype	Composite Structure Diagram
	AddStructuralFeatureValueAct	realization	Behavior Diagram
	AddVariableValueAction	specification	Any Diagram
	AnyReceiveEvent	TODO_Owner	Static Diagram
		type	

Press Next to go the next screen.


Use First Row as Header

Properties

Name	Type	Owner
ownedComment	String	Element
componentOfPack...	Component	Activity
abstract	boolean	Activity
active	boolean	Activity
leaf	boolean	Activity
name	String	Activity
owner	Element	Activity
owningPackage	Package	Activity
package	Package	Activity
readOnly	boolean	Activity
reentrant	boolean	Activity
singleExecution	boolean	Activity

CSV Data

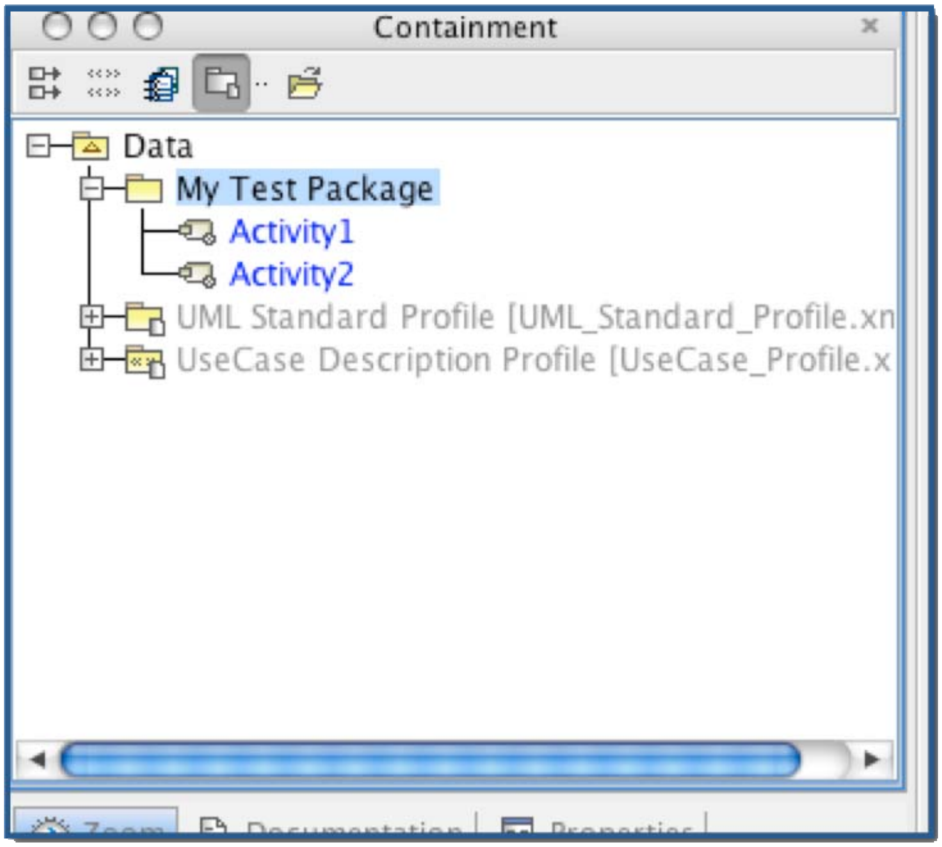
0
Activity1
Activity2

Property Maps

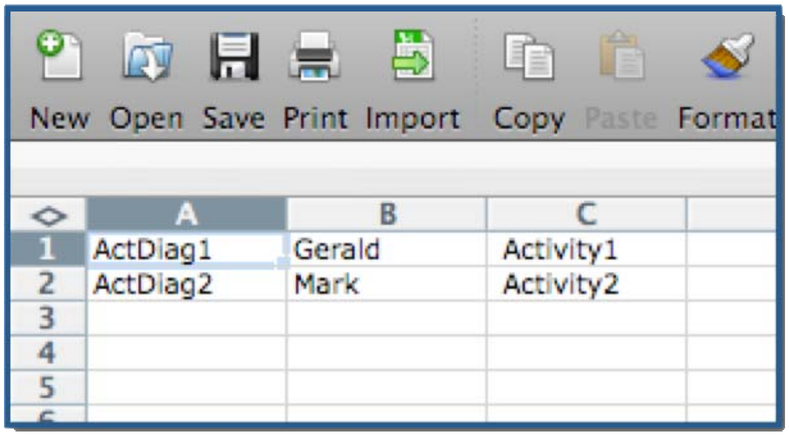
Property Name	Owner	Column Index
name	Activity	0

Key property: name

Press Finish and your containment area should look like this:

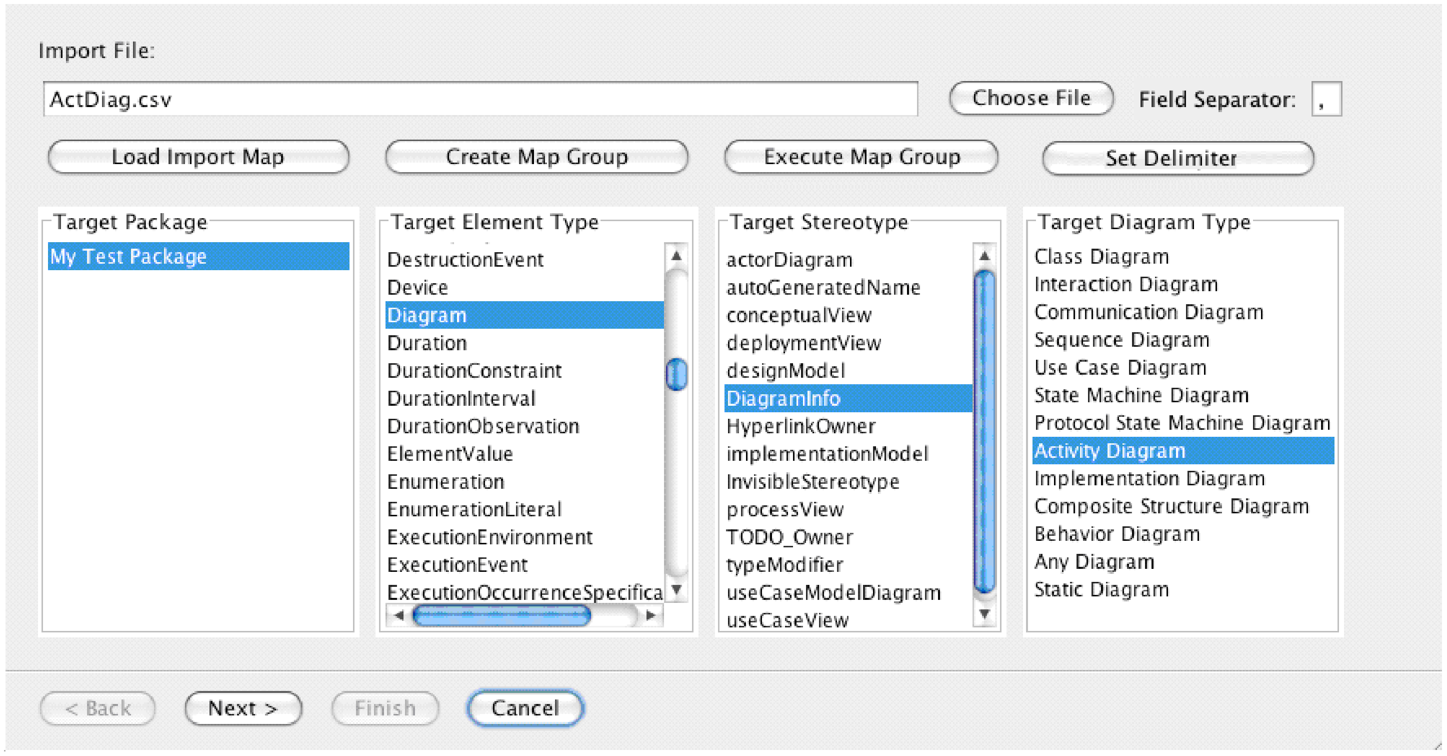

Containment
Data
My Test Package
Activity1
Activity2
UML Standard Profile [UML_Standard_Profile.xn
UseCase Description Profile [UseCase_Profile.x

Now, we'll create a couple of Activity Diagrams. As always, we start with CSV data:



	A	B	C
1	ActDiag1	Gerald	Activity1
2	ActDiag2	Mark	Activity2
3			
4			
5			

Now, we'll bring up the plug-in and import the data:

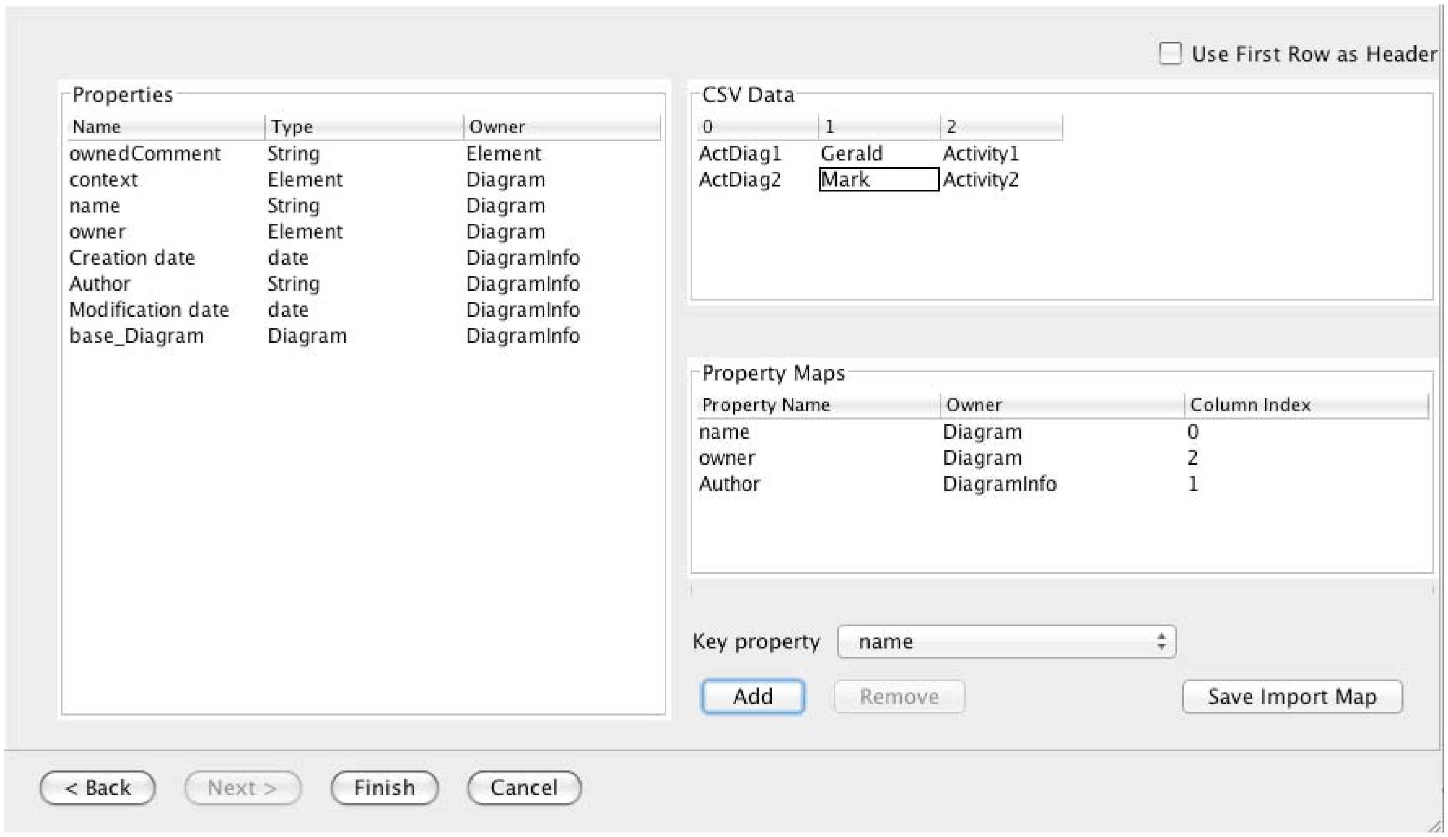


Import File: ActDiag.csv

Field Separator: ,

Target Package	Target Element Type	Target Stereotype	Target Diagram Type
My Test Package	DestructionEvent	actorDiagram	Class Diagram
	Device	autoGeneratedName	Interaction Diagram
	Diagram	conceptualView	Communication Diagram
	Duration	deploymentView	Sequence Diagram
	DurationConstraint	designModel	Use Case Diagram
	DurationInterval	DiagramInfo	State Machine Diagram
	DurationObservation	HyperlinkOwner	Protocol State Machine Diagram
	ElementValue	implementationModel	Activity Diagram
	Enumeration	InvisibleStereotype	Implementation Diagram
	EnumerationLiteral	processView	Composite Structure Diagram
	ExecutionEnvironment	TODO_Owner	Behavior Diagram
	ExecutionEvent	typeModifier	Any Diagram
	ExecutionOccurrenceSpecifica	useCaseModelDiagram	Static Diagram
		useCaseView	

Press Next to go to the next screen.



Properties

Name	Type	Owner
ownedComment	String	Element
context	Element	Diagram
name	String	Diagram
owner	Element	Diagram
Creation date	date	DiagramInfo
Author	String	DiagramInfo
Modification date	date	DiagramInfo
base_Diagram	Diagram	DiagramInfo

CSV Data

0	1	2
ActDiag1	Gerald	Activity1
ActDiag2	Mark	Activity2

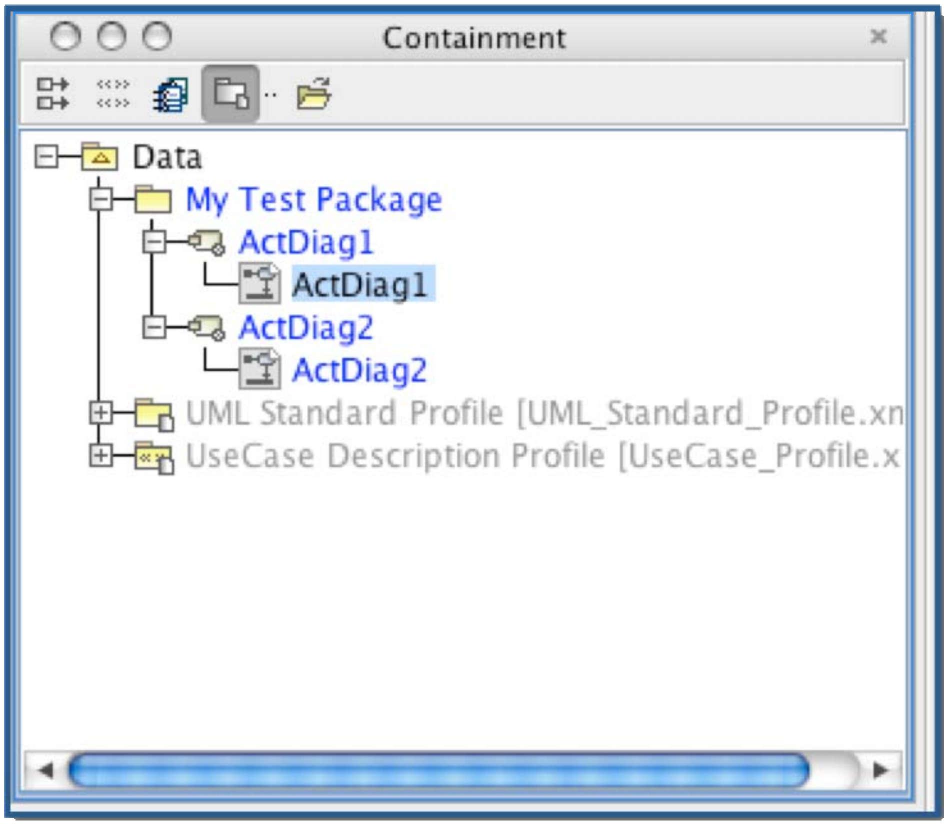
Property Maps

Property Name	Owner	Column Index
name	Diagram	0
owner	Diagram	2
Author	DiagramInfo	1

Key property: name

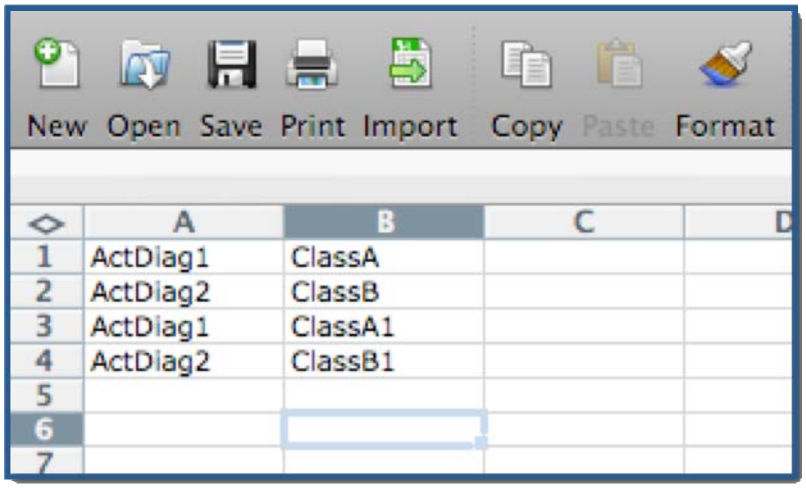
Press Finish.

MagicDraw will most likely rename your activities to match the diagram name, so your containment area will look like this:


Containment
Data
My Test Package
ActDiag1
ActDiag1
ActDiag2
ActDiag2
UML Standard Profile [UML_Standard_Profile.xn
UseCase Description Profile [UseCase_Profile.x

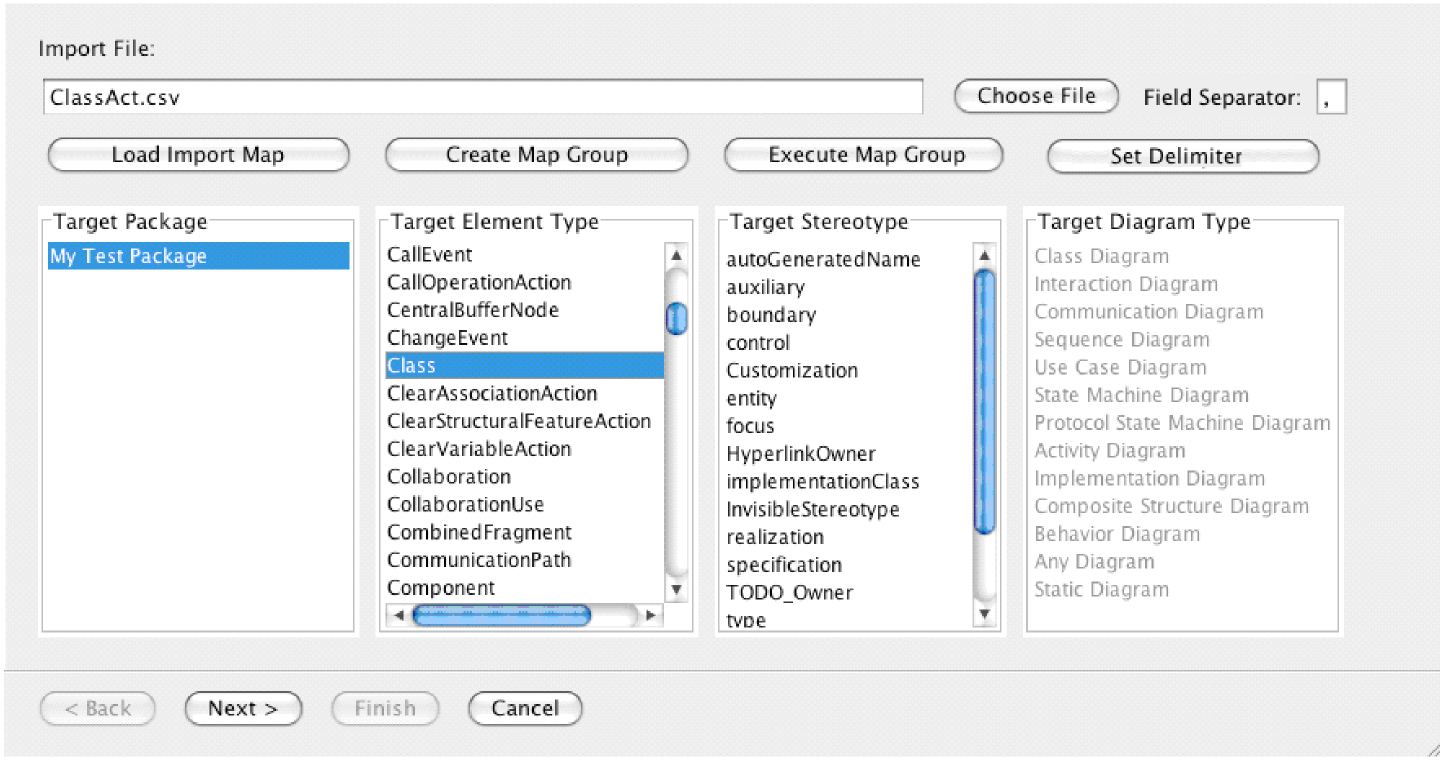
Creating Classes into Diagram

Now, let's create some classes to place on these diagrams:



	A	B	C	D
1	ActDiag1	ClassA		
2	ActDiag2	ClassB		
3	ActDiag1	ClassA1		
4	ActDiag2	ClassB1		
5				
6				
7				

Start the plug-in


Import File: ClassAct.csv

Field Separator: ,

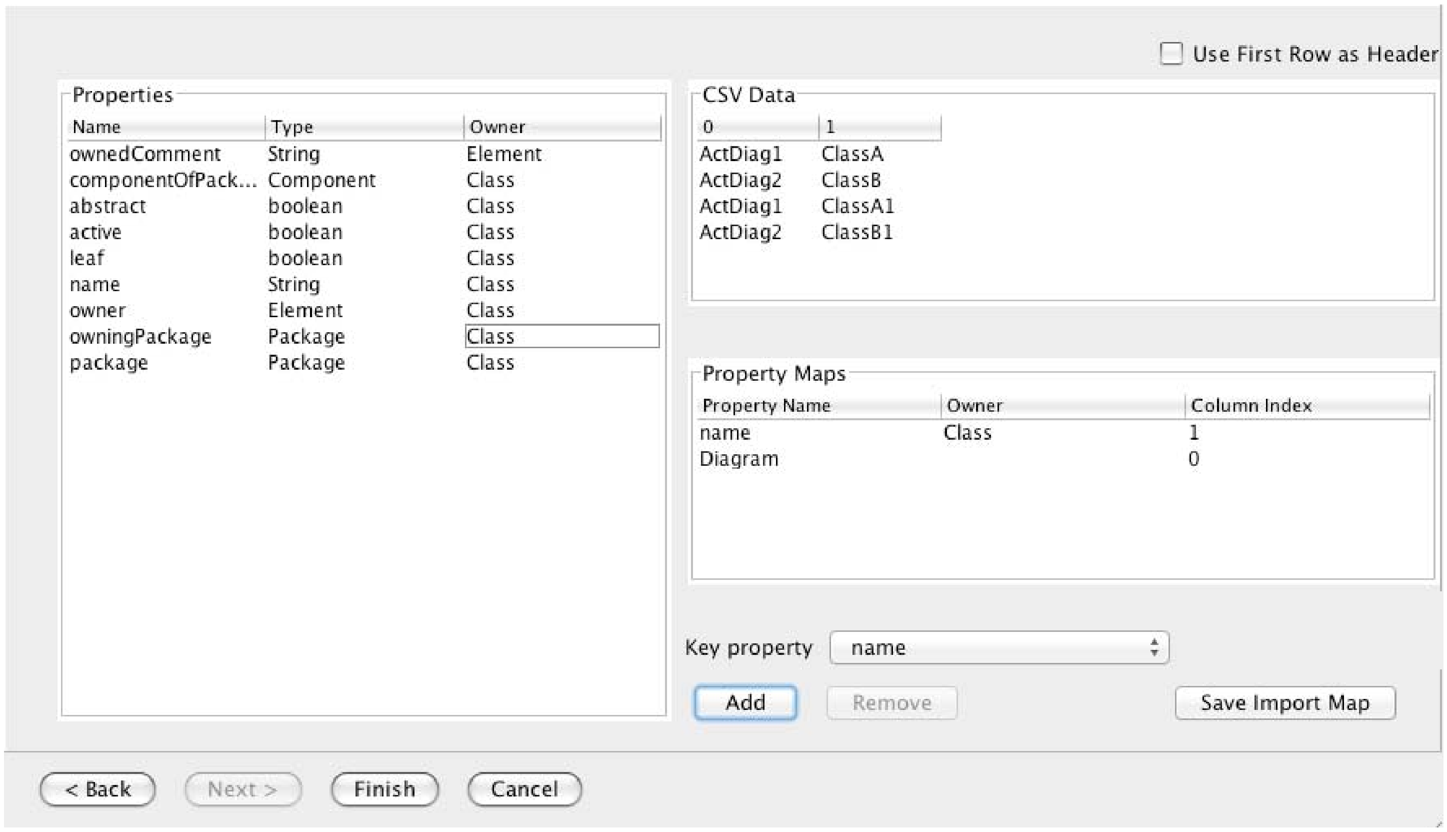
Target Package: My Test Package

Target Element Type: CallEvent, CallOperationAction, CentralBufferNode, ChangeEvent, Class, ClearAssociationAction, ClearStructuralFeatureAction, ClearVariableAction, Collaboration, CollaborationUse, CombinedFragment, CommunicationPath, Component

Target Stereotype: autoGeneratedName, auxiliary, boundary, control, Customization, entity, focus, HyperlinkOwner, implementationClass, InvisibleStereotype, realization, specification, TODO_Owner, type

Target Diagram Type: Class Diagram, Interaction Diagram, Communication Diagram, Sequence Diagram, Use Case Diagram, State Machine Diagram, Protocol State Machine Diagram, Activity Diagram, Implementation Diagram, Composite Structure Diagram, Behavior Diagram, Any Diagram, Static Diagram

Press Next to go the next screen.


Properties

Name	Type	Owner
ownedComment	String	Element
componentOfPack...	Component	Class
abstract	boolean	Class
active	boolean	Class
leaf	boolean	Class
name	String	Class
owner	Element	Class
owningPackage	Package	Class
package	Package	Class

CSV Data

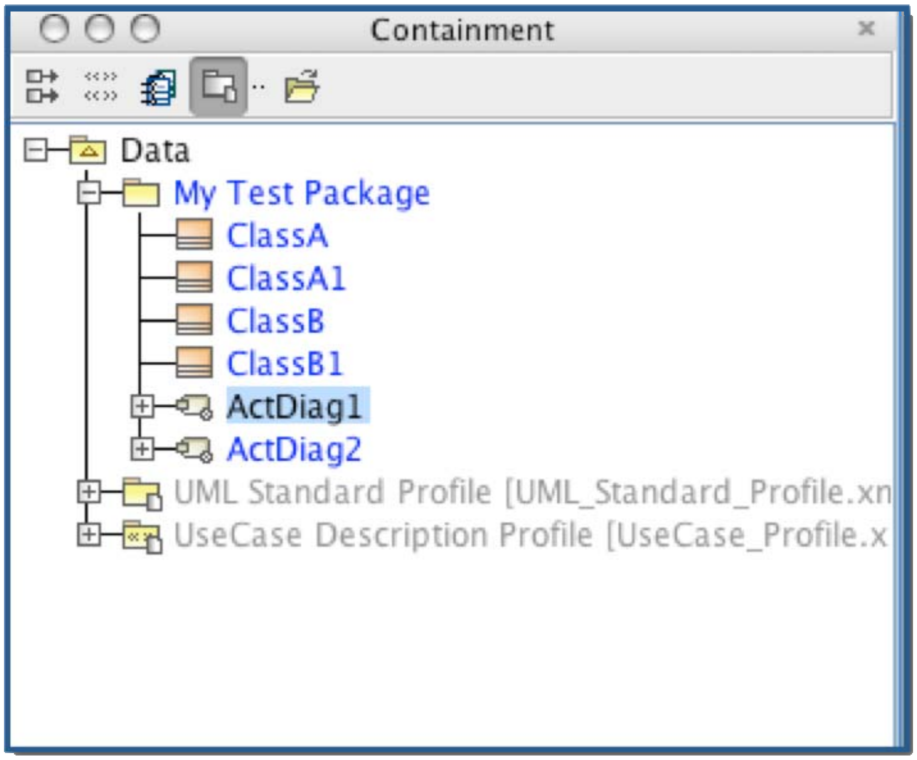
0	1
ActDiag1	ClassA
ActDiag2	ClassB
ActDiag1	ClassA1
ActDiag2	ClassB1

Property Maps

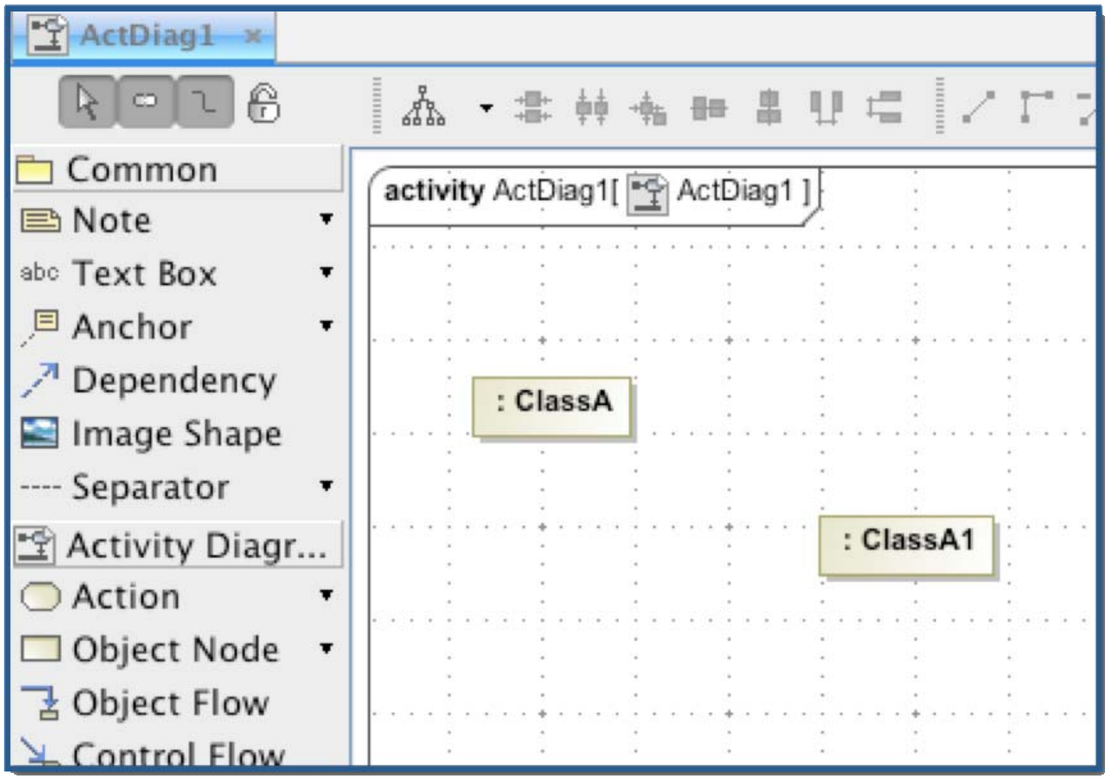
Property Name	Owner	Column Index
name	Class	1
Diagram		0

Key property: name

After pressing Finish, your containment area reflects the changes:

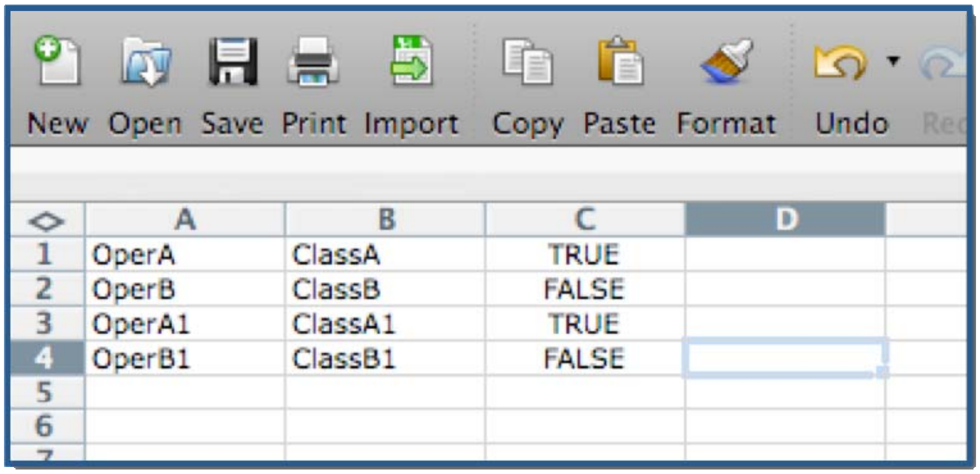

Containment
Data
My Test Package
ClassA
ClassA1
ClassB
ClassB1
ActDiag1
ActDiag2
UML Standard Profile [UML_Standard_Profile.xn
UseCase Description Profile [UseCase_Profile.x

We can also see the changes reflected on the diagrams:



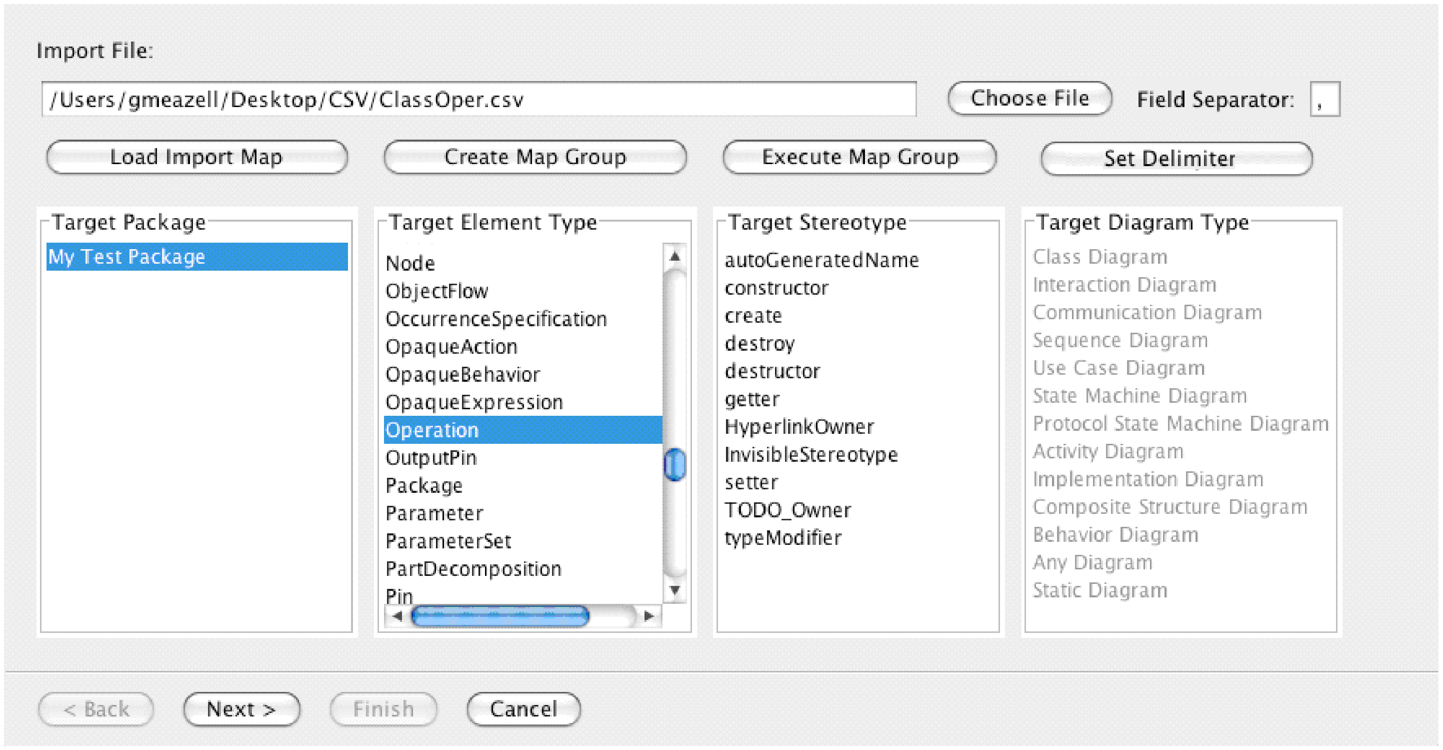
Creating Operations into Classes

Now, we will create some operations inside those classes.



	A	B	C	D
1	OperA	ClassA	TRUE	
2	OperB	ClassB	FALSE	
3	OperA1	ClassA1	TRUE	
4	OperB1	ClassB1	FALSE	
5				
6				

And import them:


Import File:

/Users/gmeazell/Desktop/CSV/ClassOper.csv

Field Separator: ,

Target Package

- My Test Package

Target Element Type

- Node
- ObjectFlow
- OccurrenceSpecification
- OpaqueAction
- OpaqueBehavior
- OpaqueExpression
- Operation
- OutputPin
- Package
- Parameter
- ParameterSet
- PartDecomposition
- Pin

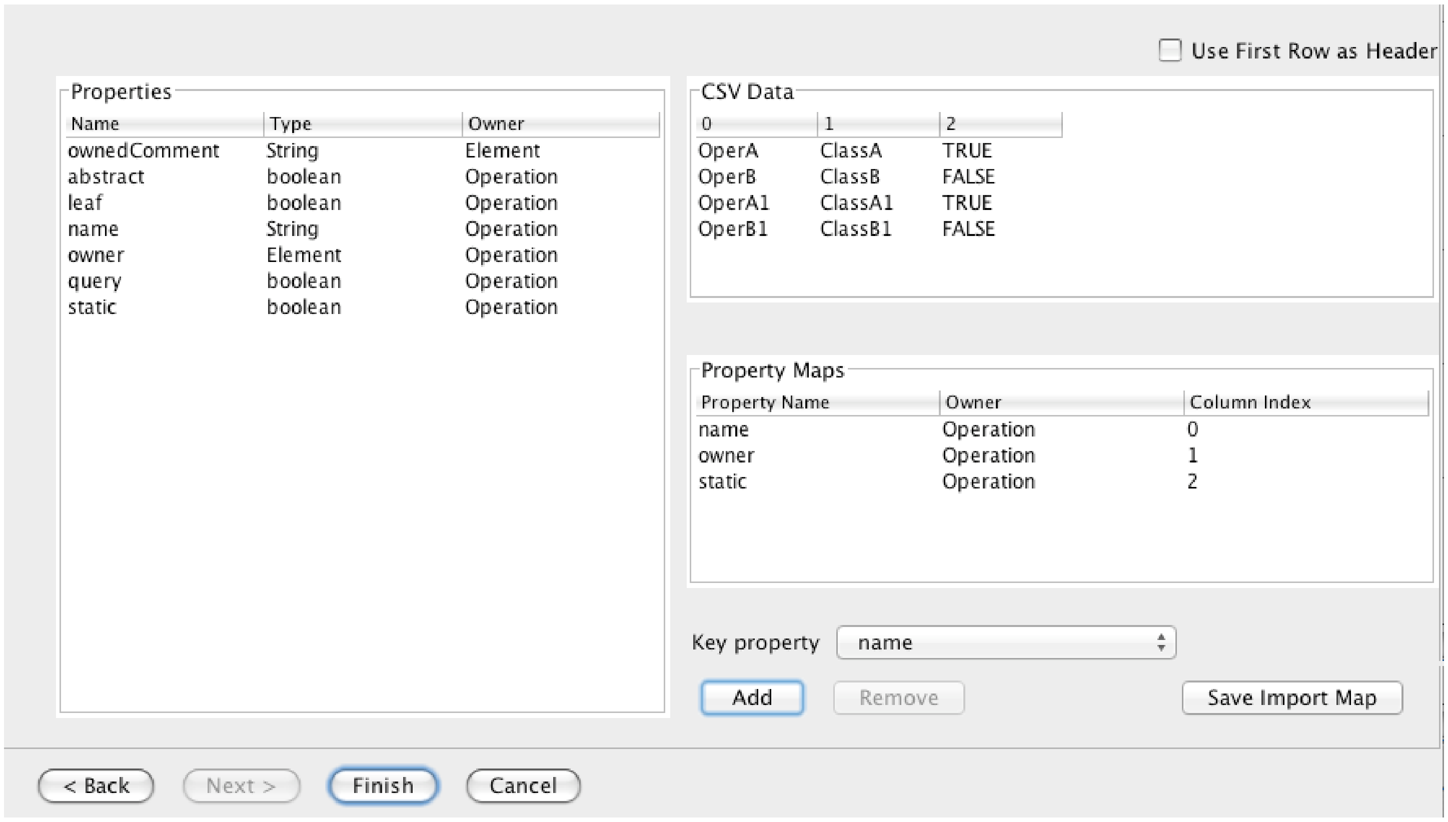
Target Stereotype

- autoGeneratedName
- constructor
- create
- destroy
- destructor
- getter
- HyperlinkOwner
- InvisibleStereotype
- setter
- TODO_Owner
- typeModifier

Target Diagram Type

- Class Diagram
- Interaction Diagram
- Communication Diagram
- Sequence Diagram
- Use Case Diagram
- State Machine Diagram
- Protocol State Machine Diagram
- Activity Diagram
- Implementation Diagram
- Composite Structure Diagram
- Behavior Diagram
- Any Diagram
- Static Diagram

Press Next to go the next screen.


Use First Row as Header

Properties

Name	Type	Owner
ownedComment	String	Element
abstract	boolean	Operation
leaf	boolean	Operation
name	String	Operation
owner	Element	Operation
query	boolean	Operation
static	boolean	Operation

CSV Data

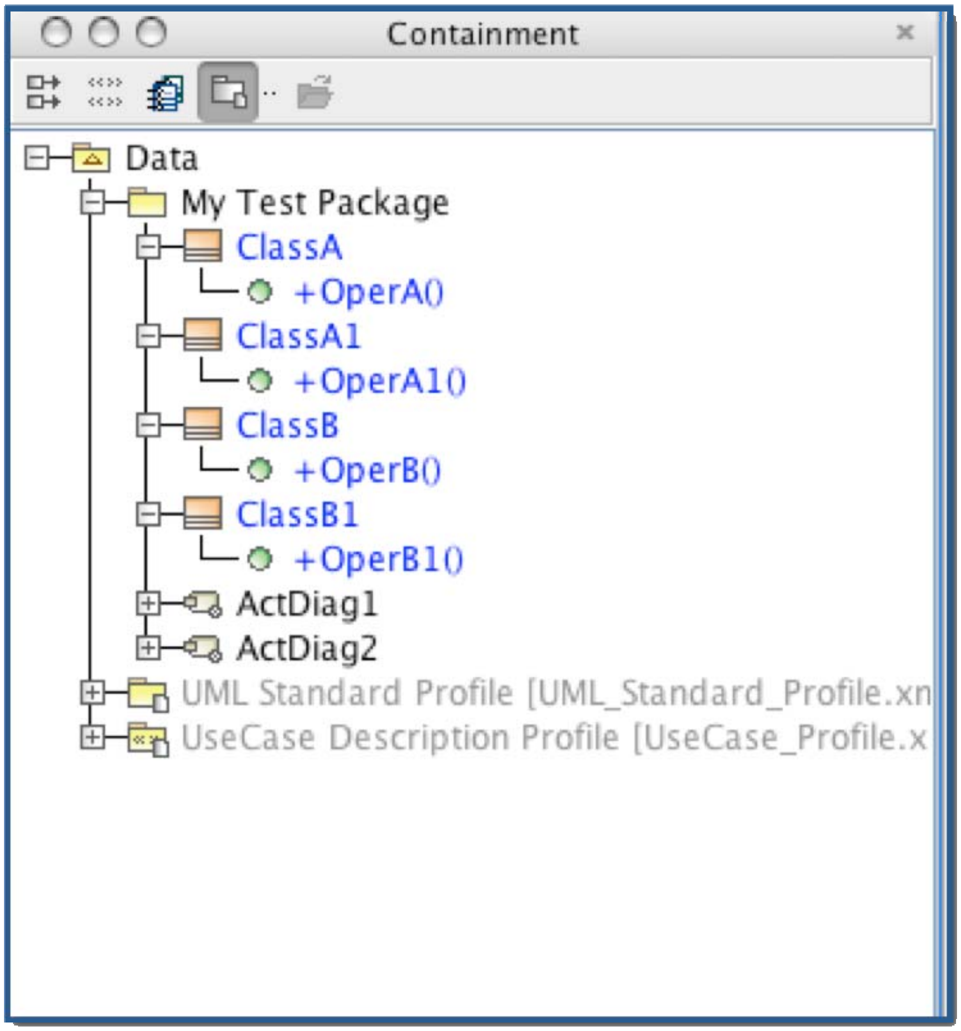
0	1	2
OperA	ClassA	TRUE
OperB	ClassB	FALSE
OperA1	ClassA1	TRUE
OperB1	ClassB1	FALSE

Property Maps

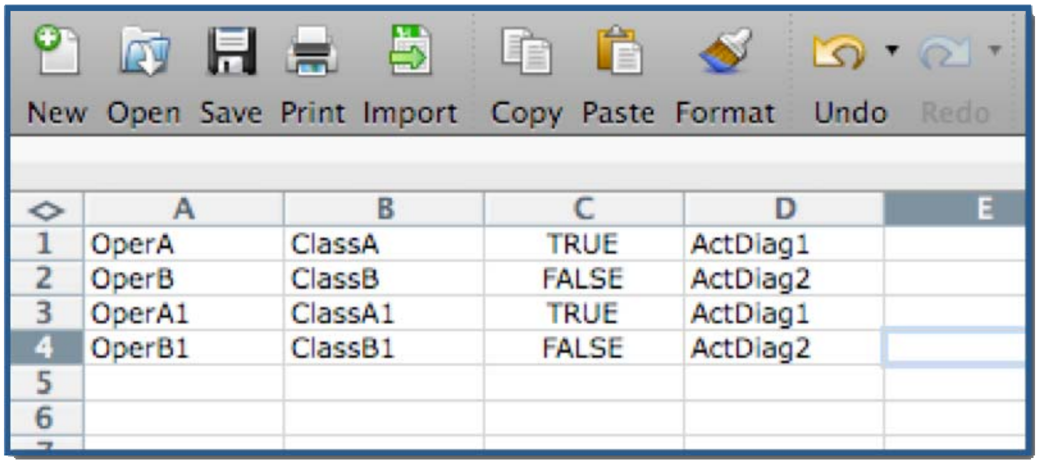
Property Name	Owner	Column Index
name	Operation	0
owner	Operation	1
static	Operation	2

Key property: name

Press Finish and we see the changes reflected in the containment area:

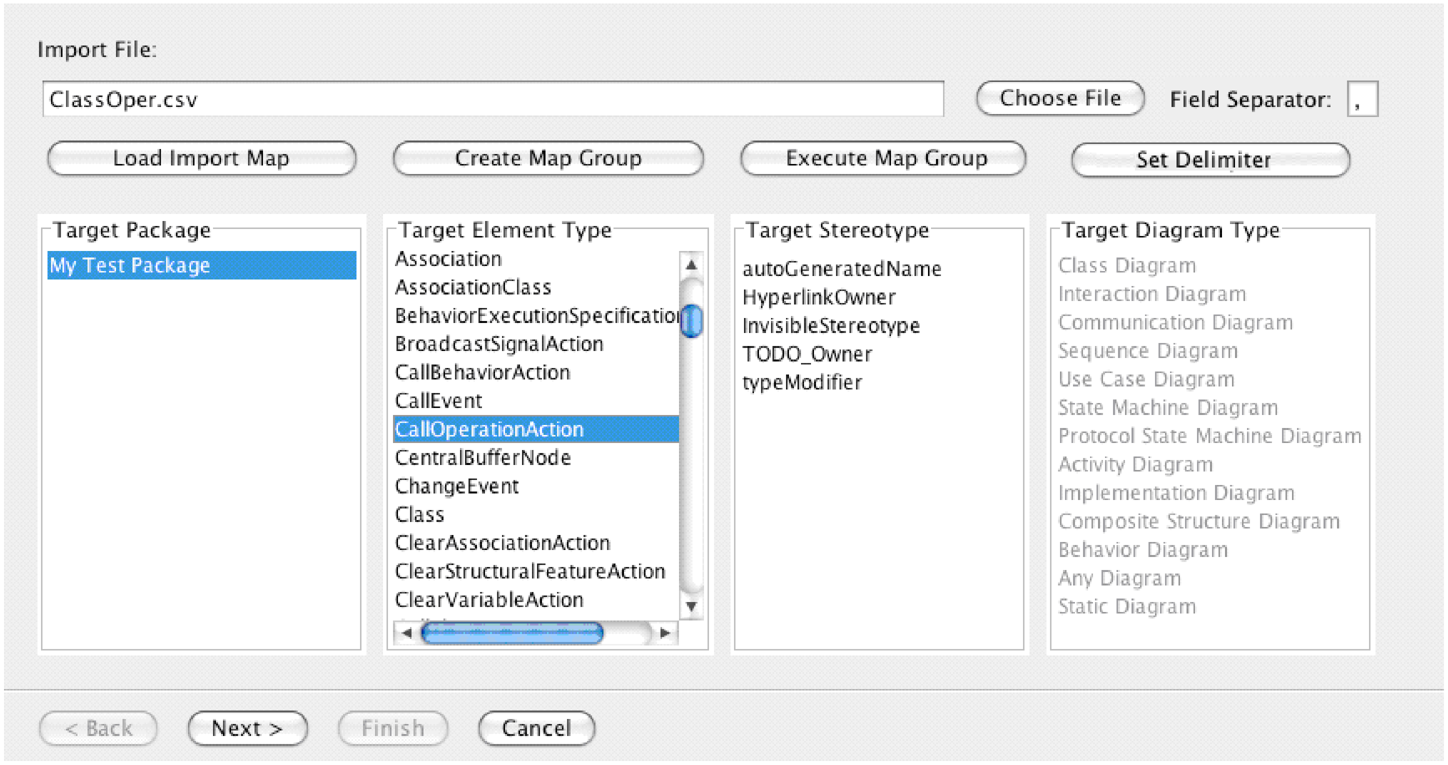


Now, we'll put these Operations on the diagram:



	A	B	C	D	E
1	OperA	ClassA	TRUE	ActDiag1	
2	OperB	ClassB	FALSE	ActDiag2	
3	OperA1	ClassA1	TRUE	ActDiag1	
4	OperB1	ClassB1	FALSE	ActDiag2	
5					
6					

Start the plug-in.



Import File: ClassOper.csv

Field Separator: ,

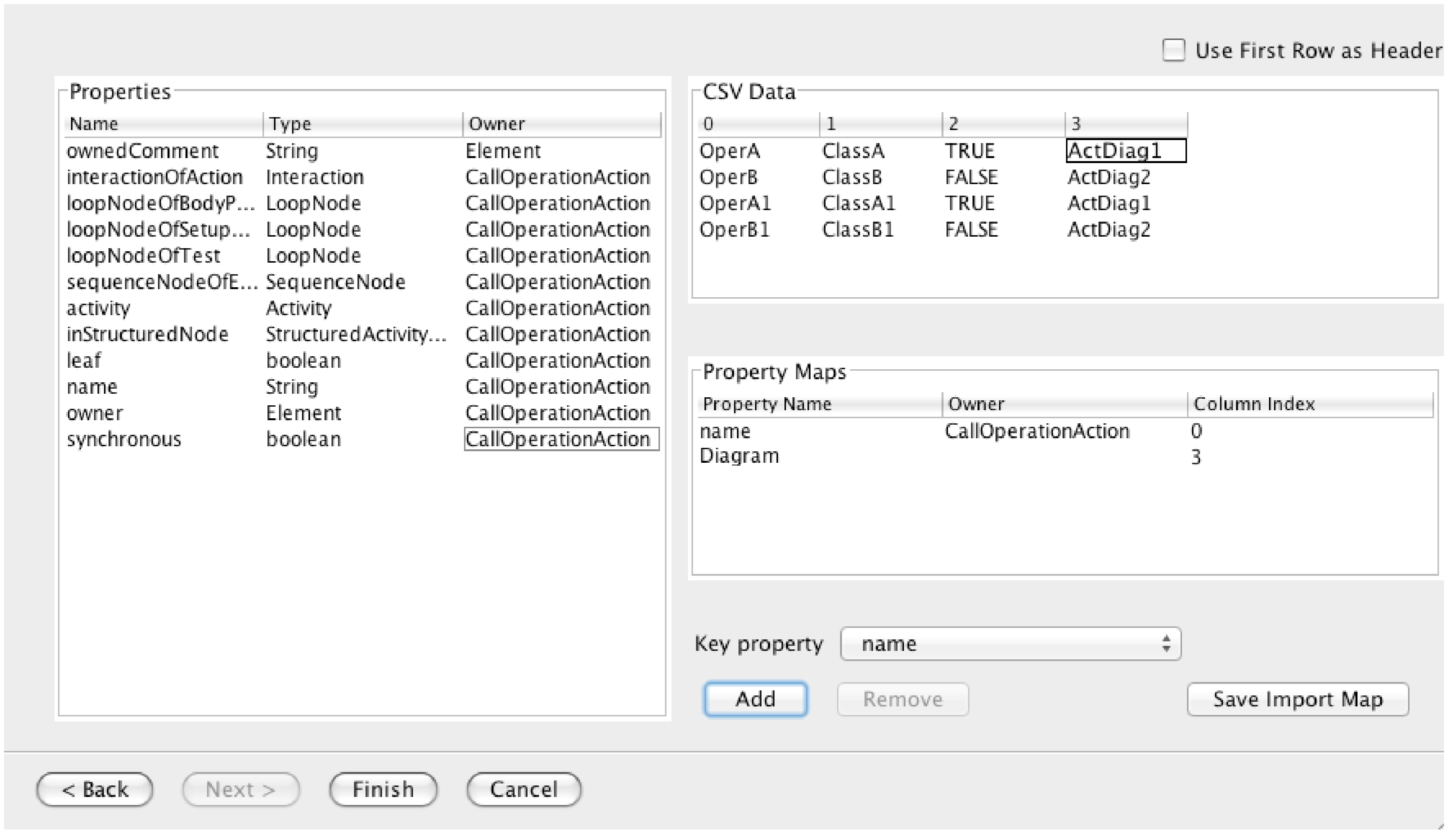
Target Package:
- My Test Package

Target Element Type:
- Association
- AssociationClass
- BehaviorExecutionSpecificatio
- BroadcastSignalAction
- CallBehaviorAction
- CallEvent
- CallOperationAction
- CentralBufferNode
- ChangeEvent
- Class
- ClearAssociationAction
- ClearStructuralFeatureAction
- ClearVariableAction

Target Stereotype:
- autoGeneratedName
- HyperlinkOwner
- InvisibleStereotype
- TODO_Owner
- typeModifier

Target Diagram Type:
- Class Diagram
- Interaction Diagram
- Communication Diagram
- Sequence Diagram
- Use Case Diagram
- State Machine Diagram
- Protocol State Machine Diagram
- Activity Diagram
- Implementation Diagram
- Composite Structure Diagram
- Behavior Diagram
- Any Diagram
- Static Diagram

Press Next to go the next screen.



Use First Row as Header

Properties

Name	Type	Owner
ownedComment	String	Element
interactionOfAction	Interaction	CallOperationAction
loopNodeOfBodyP...	LoopNode	CallOperationAction
loopNodeOfSetup...	LoopNode	CallOperationAction
loopNodeOfTest	LoopNode	CallOperationAction
sequenceNodeOfE...	SequenceNode	CallOperationAction
activity	Activity	CallOperationAction
inStructuredNode	StructuredActivity...	CallOperationAction
leaf	boolean	CallOperationAction
name	String	CallOperationAction
owner	Element	CallOperationAction
synchronous	boolean	CallOperationAction

CSV Data

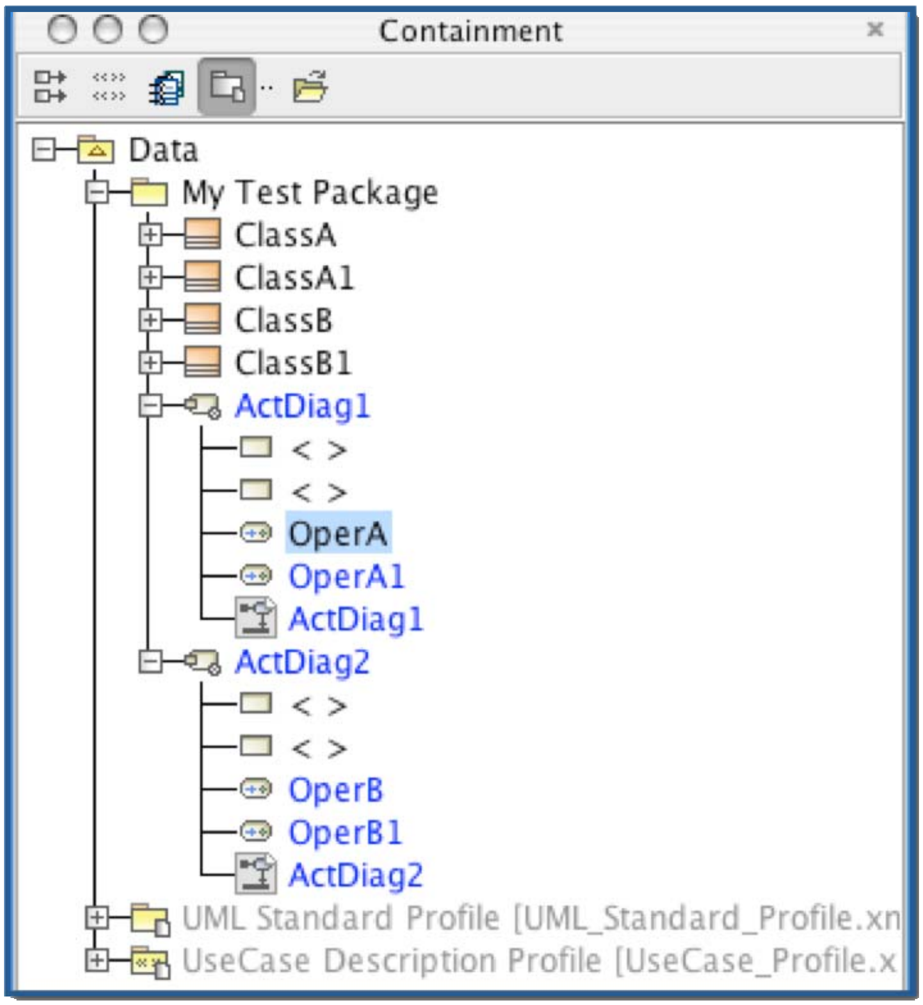
0	1	2	3
OperA	ClassA	TRUE	ActDiag1
OperB	ClassB	FALSE	ActDiag2
OperA1	ClassA1	TRUE	ActDiag1
OperB1	ClassB1	FALSE	ActDiag2

Property Maps

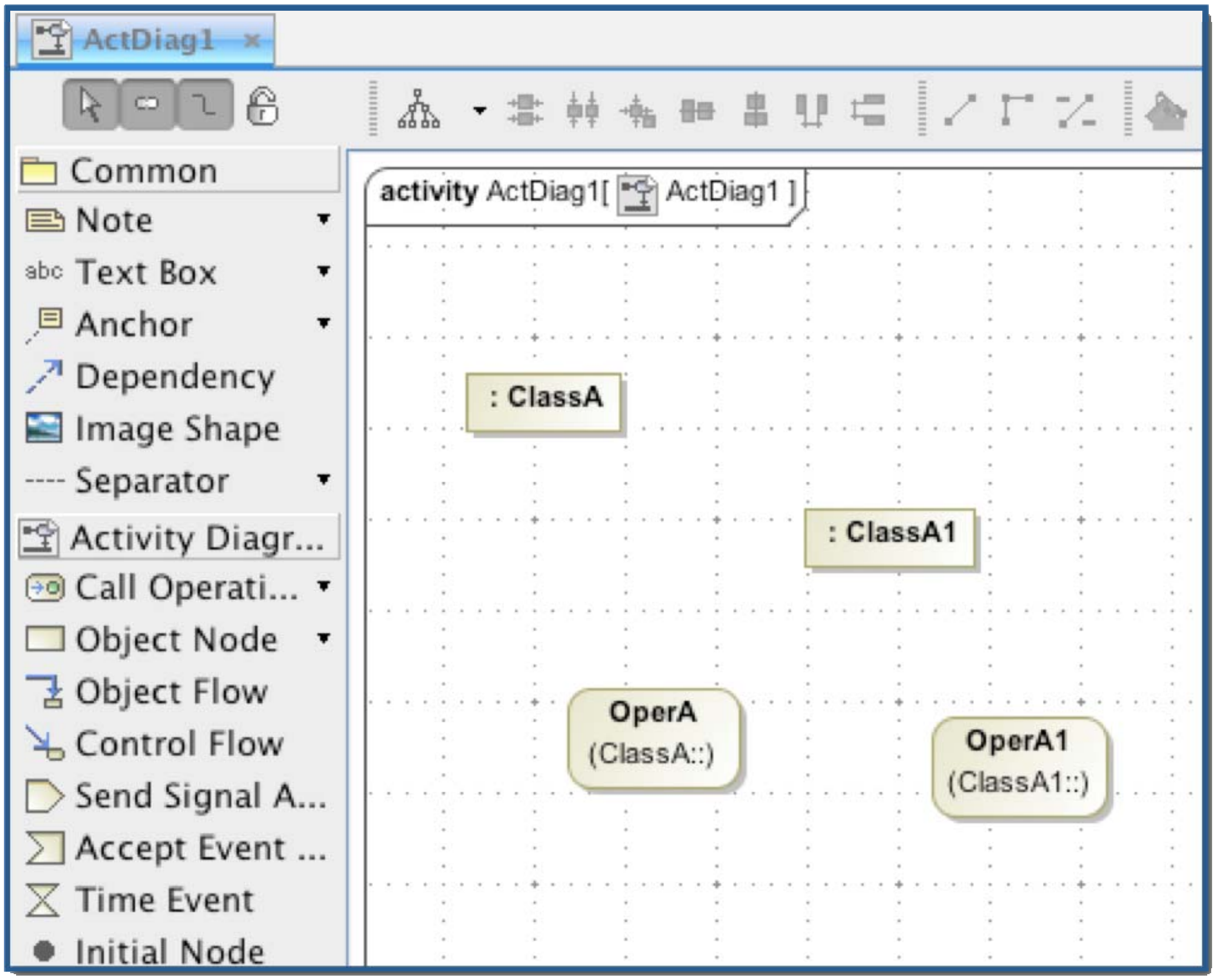
Property Name	Owner	Column Index
name	CallOperationAction	0
Diagram		3

Key property: name

After pressing Finish, the containment area gets updated...

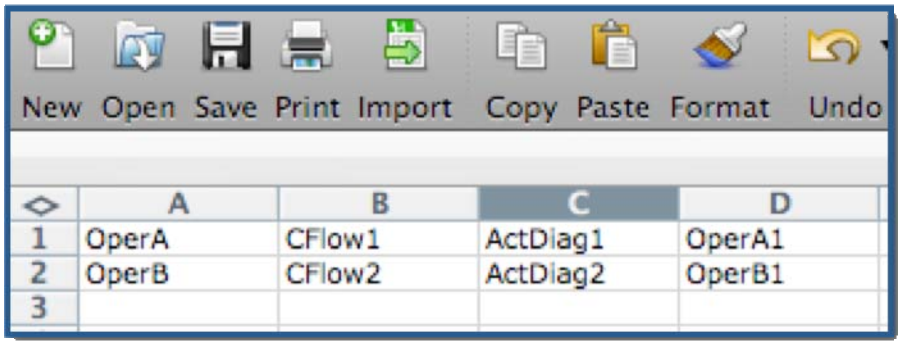


...as well as the diagrams:



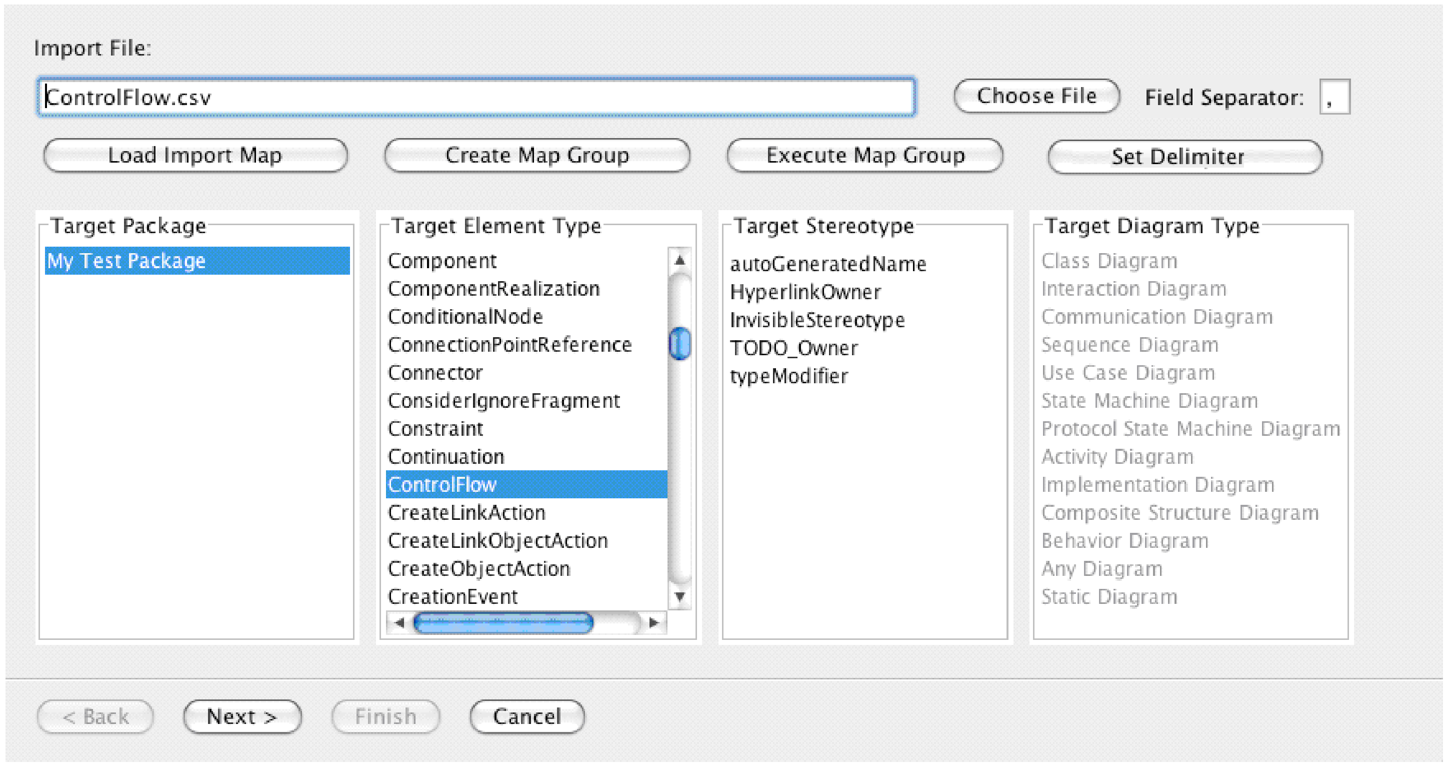
Creating Control Flows between Operations

Now, we'll tie these operations together with some control flows:



	A	B	C	D
1	OperA	CFlow1	ActDiag1	OperA1
2	OperB	CFlow2	ActDiag2	OperB1
3				

Start the plug-in.


Import File: ControlFlow.csv

Field Separator: ,

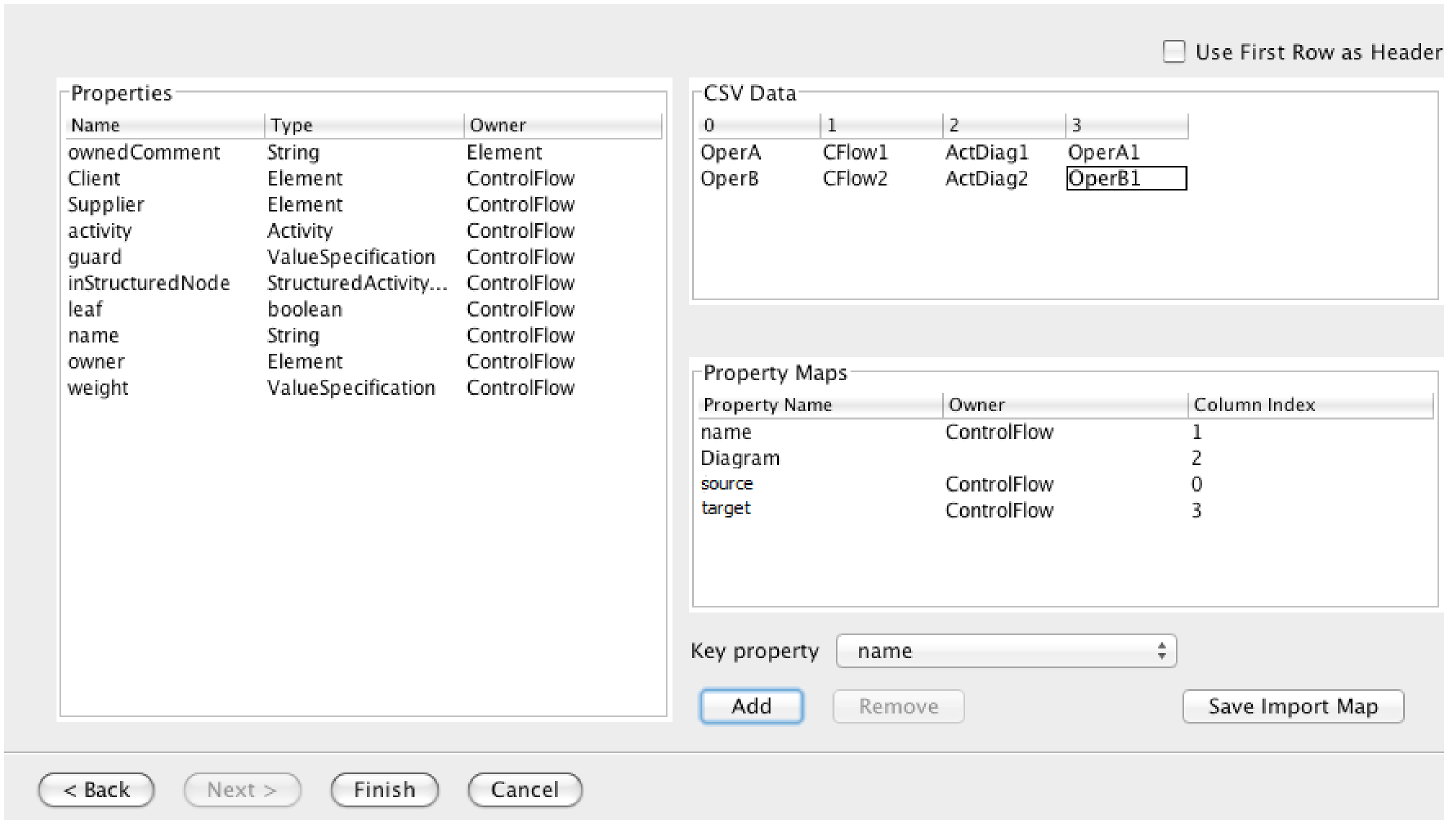
Target Package
- My Test Package

Target Element Type
- Component
- ComponentRealization
- ConditionalNode
- ConnectionPointReference
- Connector
- ConsiderIgnoreFragment
- Constraint
- Continuation
- ControlFlow
- CreateLinkAction
- CreateLinkObjectAction
- CreateObjectAction
- CreationEvent

Target Stereotype
- autoGeneratedName
- HyperlinkOwner
- InvisibleStereotype
- TODO_Owner
- typeModifier

Target Diagram Type
- Class Diagram
- Interaction Diagram
- Communication Diagram
- Sequence Diagram
- Use Case Diagram
- State Machine Diagram
- Protocol State Machine Diagram
- Activity Diagram
- Implementation Diagram
- Composite Structure Diagram
- Behavior Diagram
- Any Diagram
- Static Diagram

Press Next to go the next screen.



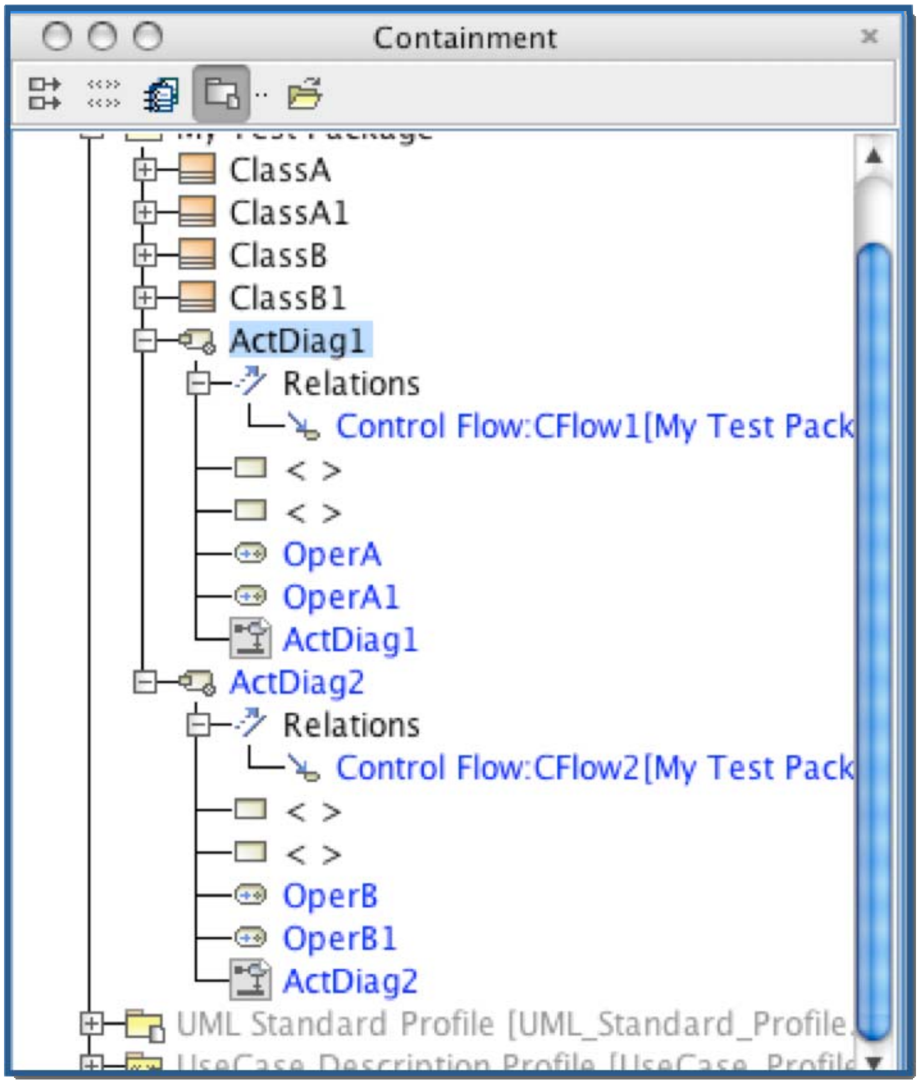
Name	Type	Owner
ownedComment	String	Element
Client	Element	ControlFlow
Supplier	Element	ControlFlow
activity	Activity	ControlFlow
guard	ValueSpecification	ControlFlow
inStructuredNode	StructuredActivity...	ControlFlow
leaf	boolean	ControlFlow
name	String	ControlFlow
owner	Element	ControlFlow
weight	ValueSpecification	ControlFlow

0	1	2	3
OperA	CFlow1	ActDiag1	OperA1
OperB	CFlow2	ActDiag2	OperB1

Property Name	Owner	Column Index
name	ControlFlow	1
Diagram		2
source	ControlFlow	0
target	ControlFlow	3

Key property: name

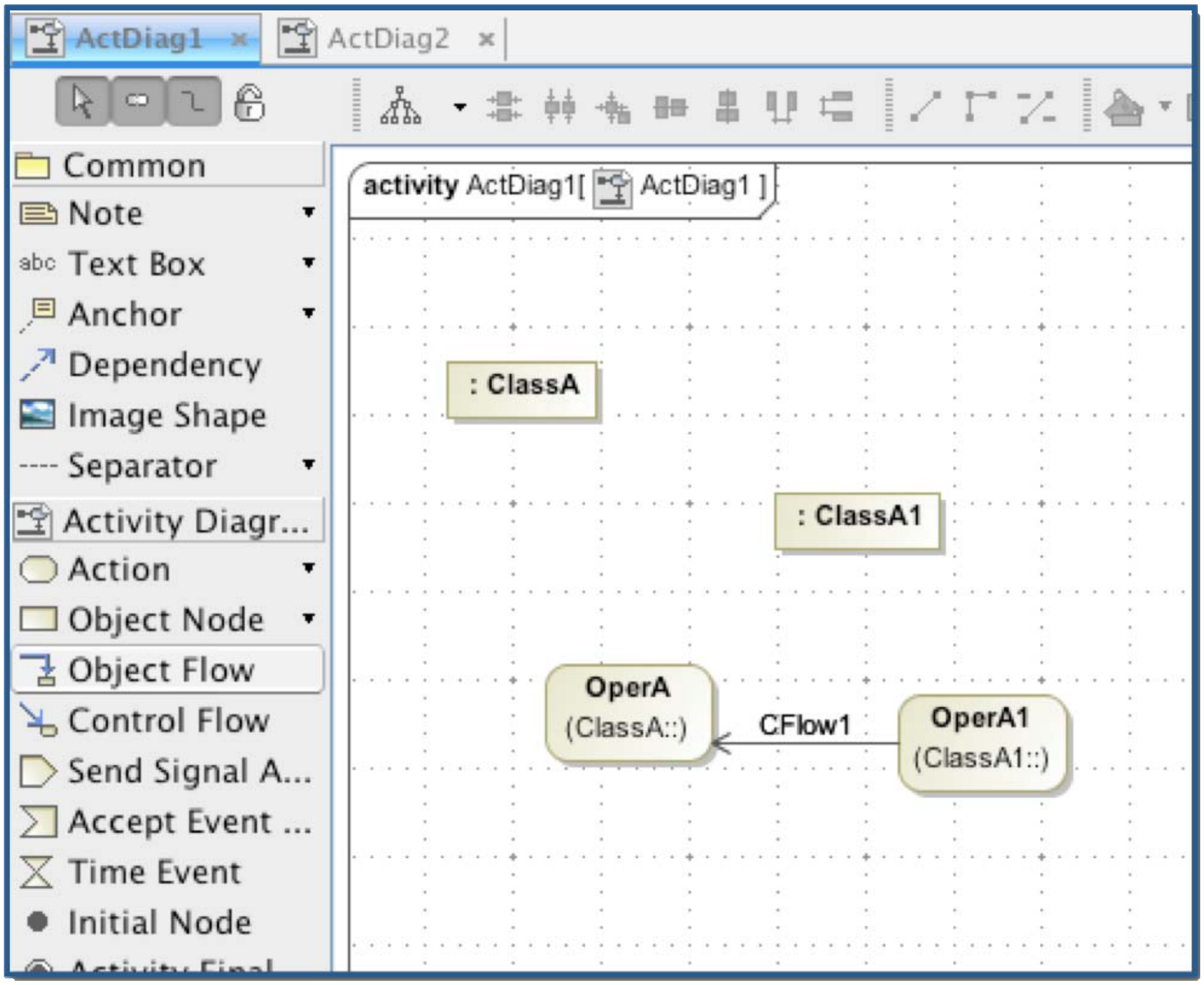
After pressing Finish, the changes are reflected in the containment area:



Containment

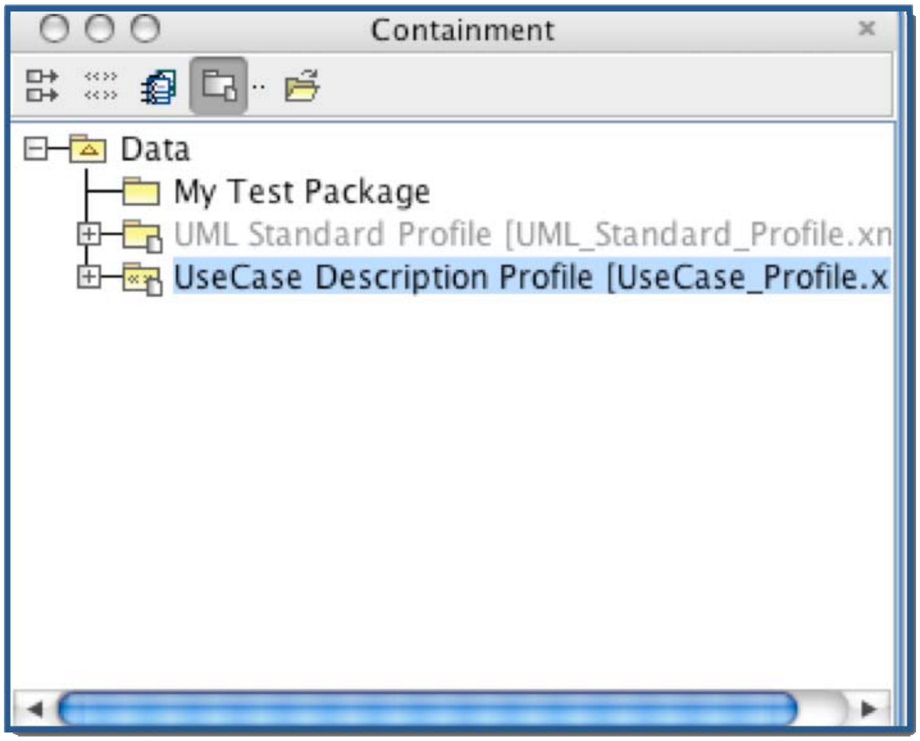
ClassA
ClassA1
ClassB
ClassB1
ActDiag1
Relations
Control Flow:CFlow1[My Test Pack
< >
< >
OperA
OperA1
ActDiag1
ActDiag2
Relations
Control Flow:CFlow2[My Test Pack
< >
< >
OperB
OperB1
ActDiag2
UML Standard Profile [UML_Standard_Profile.
UseCase Description Profile [UseCase_Profile

And on the diagrams:

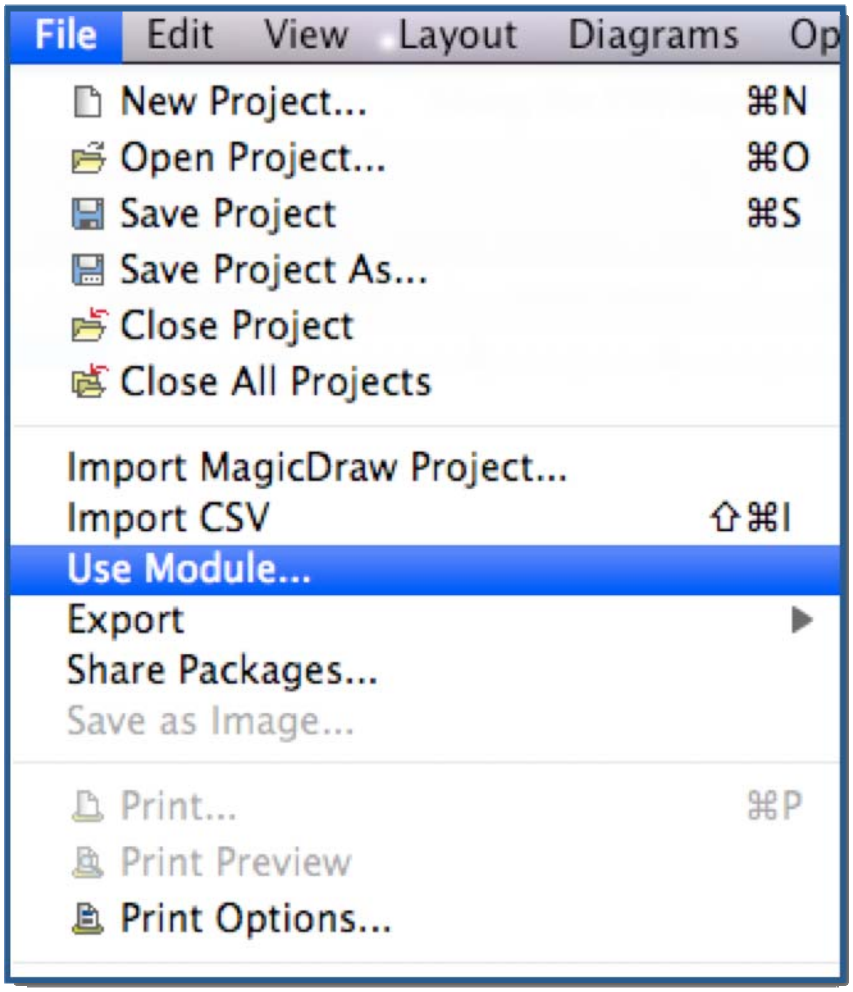


Creating and Populating DoDAF Diagrams

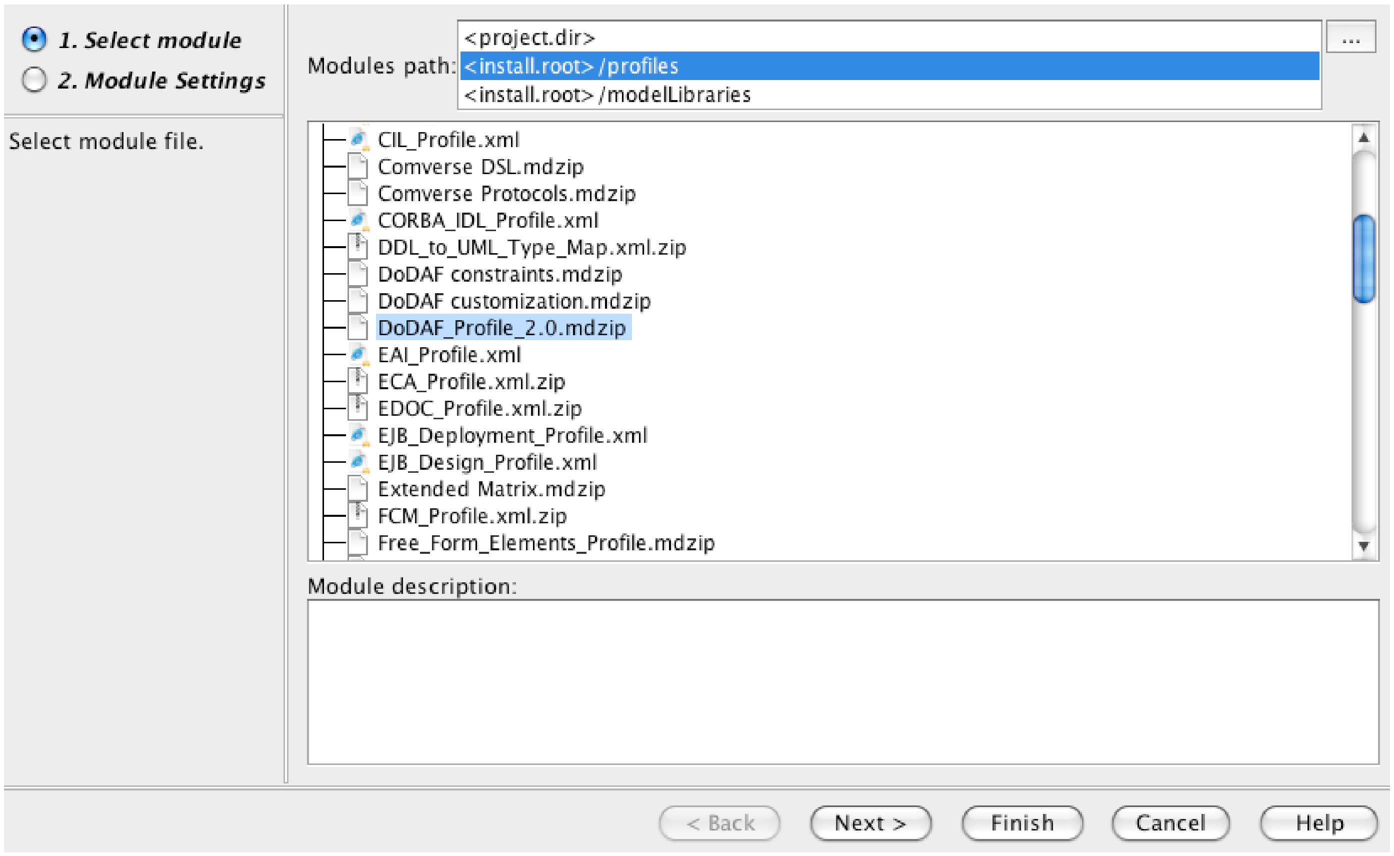
First, we’ll start with a fresh containment area:


Containment
Data
My Test Package
UML Standard Profile [UML_Standard_Profile.xn
UseCase Description Profile [UseCase_Profile.x

Now, we'll select to use the DoDAF module:



Press Next to go the next


[✓ 1. Select module] 2. Module Settings

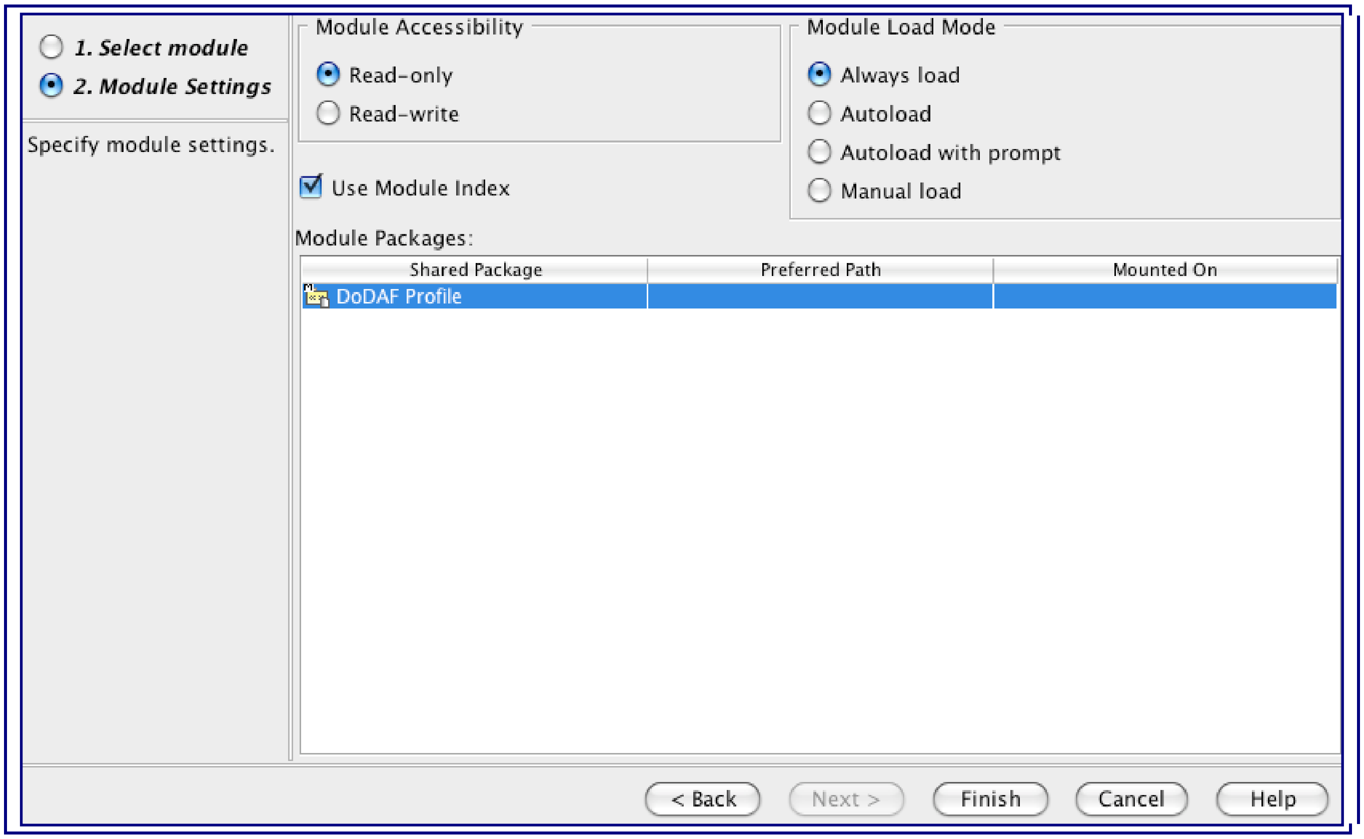
Select module file.

Modules path: <project.dir> / [✓ <install.root>/profiles] / <install.root>/modelLibraries

- CIL_Profile.xml
- Comverse DSL.mdzip
- Comverse Protocols.mdzip
- CORBA_IDL_Profile.xml
- DDL_to_UML_Type_Map.xml.zip
- DoDAF constraints.mdzip
- DoDAF customization.mdzip
- [✓ DoDAF_Profile_2.0.mdzip]
- EAI_Profile.xml
- ECA_Profile.xml.zip
- EDOC_Profile.xml.zip
- EJB_Deployment_Profile.xml
- EJB_Design_Profile.xml
- Extended Matrix.mdzip
- FCM_Profile.xml.zip
- Free_Form_Elements_Profile.mdzip

Module description:

screen.


1. Select module [✓ 2. Module Settings]

Specify module settings.

Module Accessibility: [✓ Read-only] Read-write

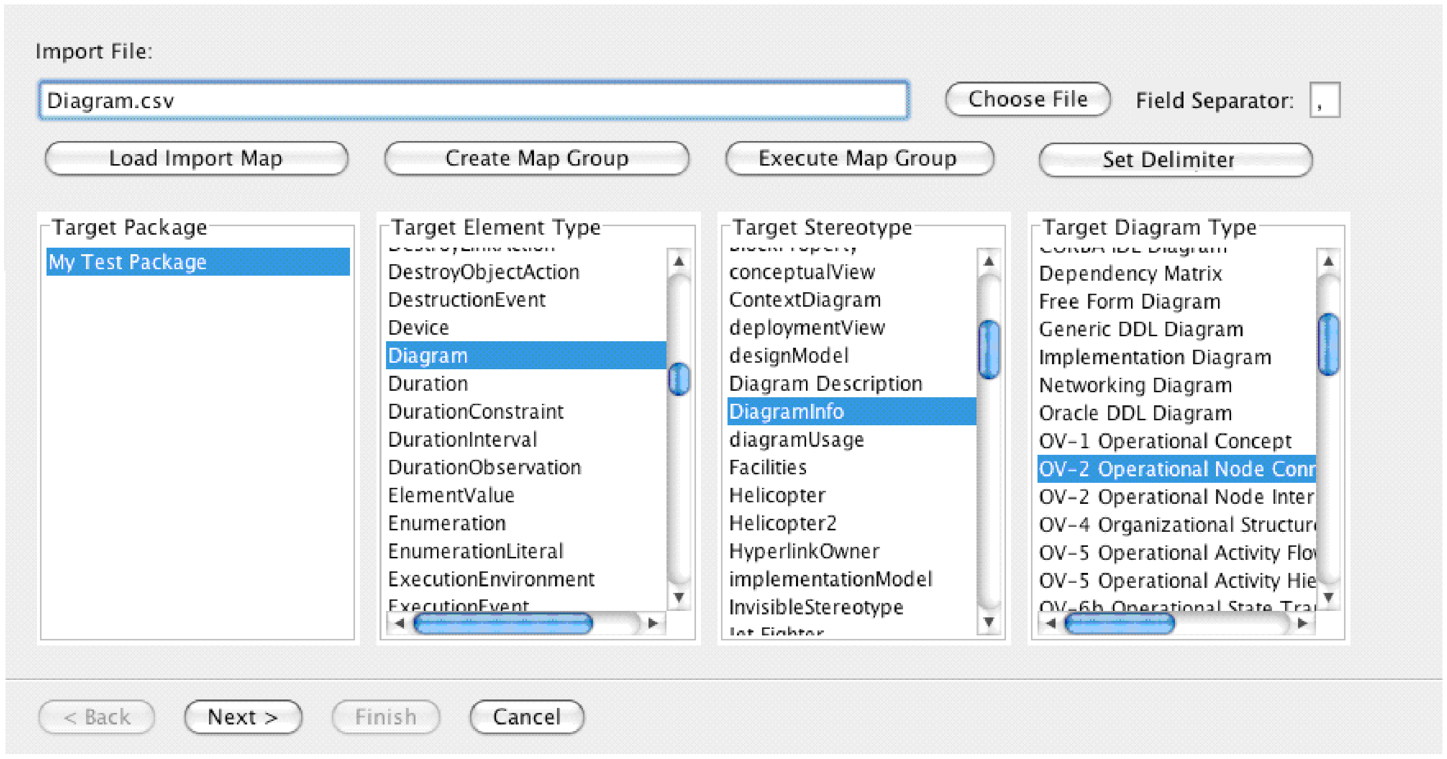
[✓ Use Module Index]

Module Load Mode: [✓ Always load] Autoload / Autoload with prompt / Manual load

Module Packages:

Shared Package	Preferred Path	Mounted On
DoDAF Profile		

Press Finish to load the profile. With the profile loaded, we will now be able to build diagrams. We'll use the class diagram example, but make a few simple changes. Select the OV-2 diagram type when importing the diagrams. Bring up the plug-in.


Import File: Diagram.csv

Field Separator: ,

Target Package: My Test Package

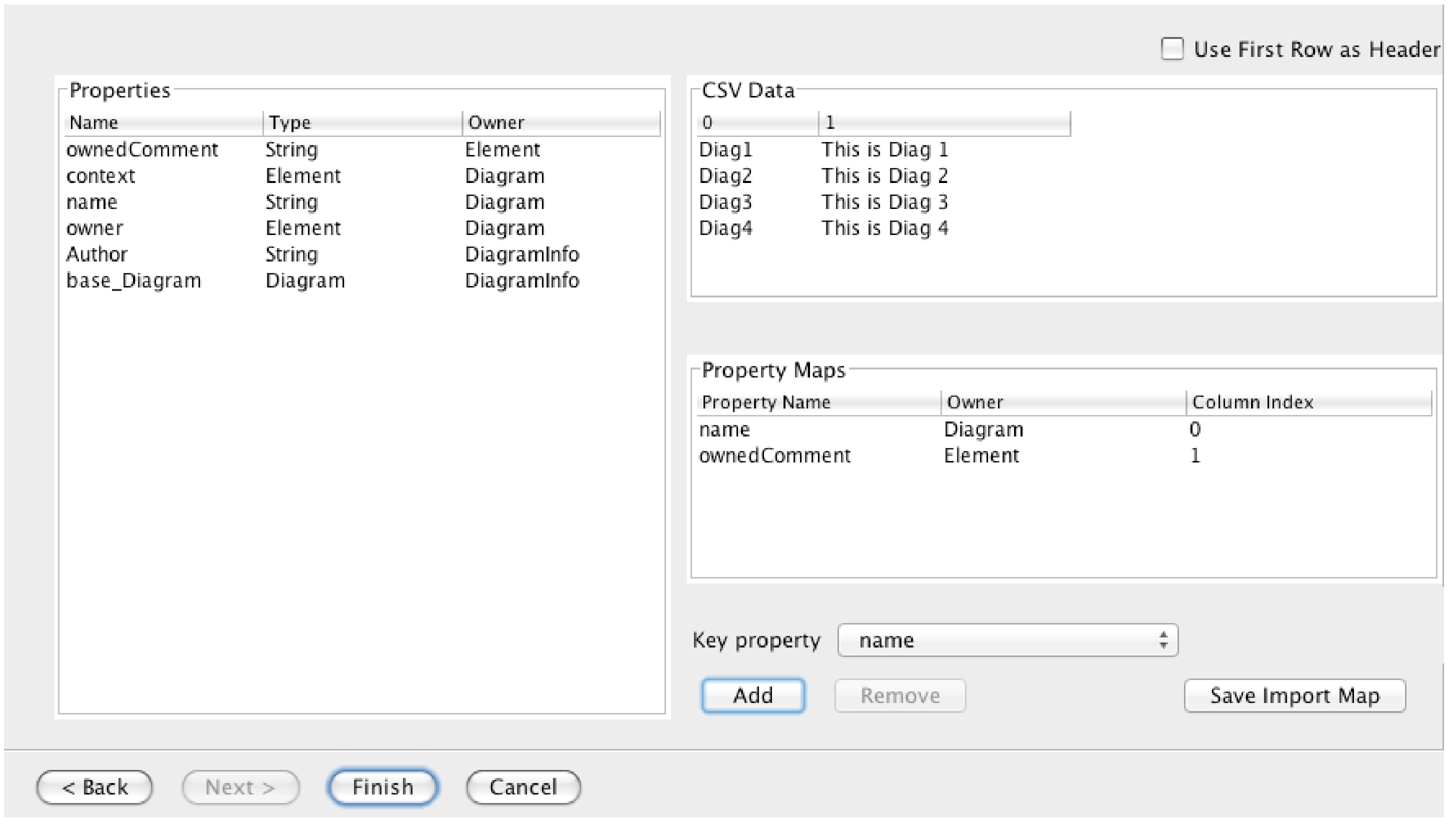
Target Element Type: DestroyObjectAction, DestructionEvent, Device, Diagram, Duration, DurationConstraint, DurationInterval, DurationObservation, ElementValue, Enumeration, EnumerationLiteral, ExecutionEnvironment, ExecutionEvent

Target Stereotype: conceptualView, ContextDiagram, deploymentView, designModel, Diagram Description, DiagramInfo, diagramUsage, Facilities, Helicopter, Helicopter2, HyperlinkOwner, implementationModel, InvisibleStereotype, Jet Fighter

Target Diagram Type: Dependency Matrix, Free Form Diagram, Generic DDL Diagram, Implementation Diagram, Networking Diagram, Oracle DDL Diagram, OV-1 Operational Concept, OV-2 Operational Node Conn, OV-2 Operational Node Inter, OV-4 Organizational Structur, OV-5 Operational Activity Flo, OV-5 Operational Activity Hie, OV-6b Operational State Tra

Selected: Target Package "My Test Package"; Target Element Type "Diagram"; Target Stereotype "DiagramInfo"; Target Diagram Type "OV-2 Operational Node Conn".

Press Next to go the next screen.



☐ Use First Row as Header

Properties

Name	Type	Owner
ownedComment	String	Element
context	Element	Diagram
name	String	Diagram
owner	Element	Diagram
Author	String	DiagramInfo
base_Diagram	Diagram	DiagramInfo

CSV Data

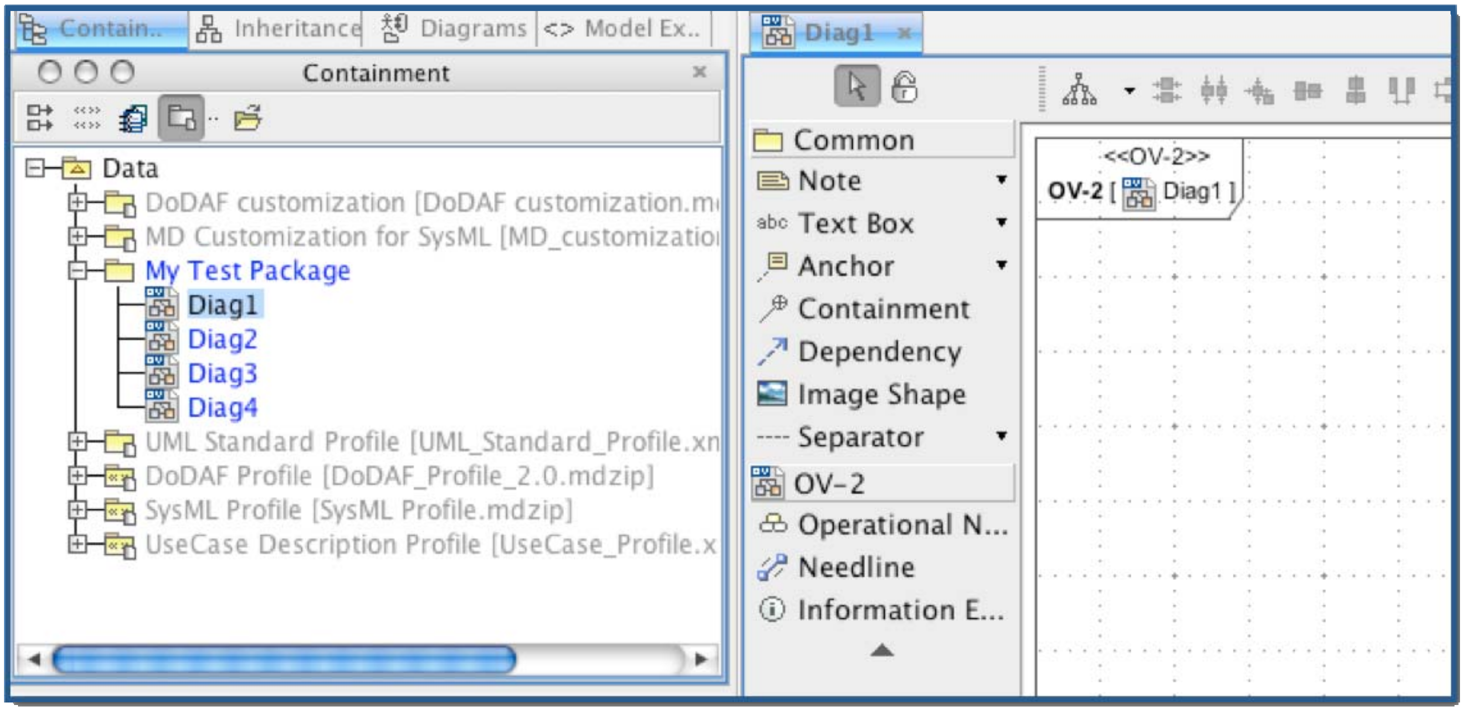
0	1
Diag1	This is Diag 1
Diag2	This is Diag 2
Diag3	This is Diag 3
Diag4	This is Diag 4

Property Maps

Property Name	Owner	Column Index
name	Diagram	0
ownedComment	Element	1

Key property: name

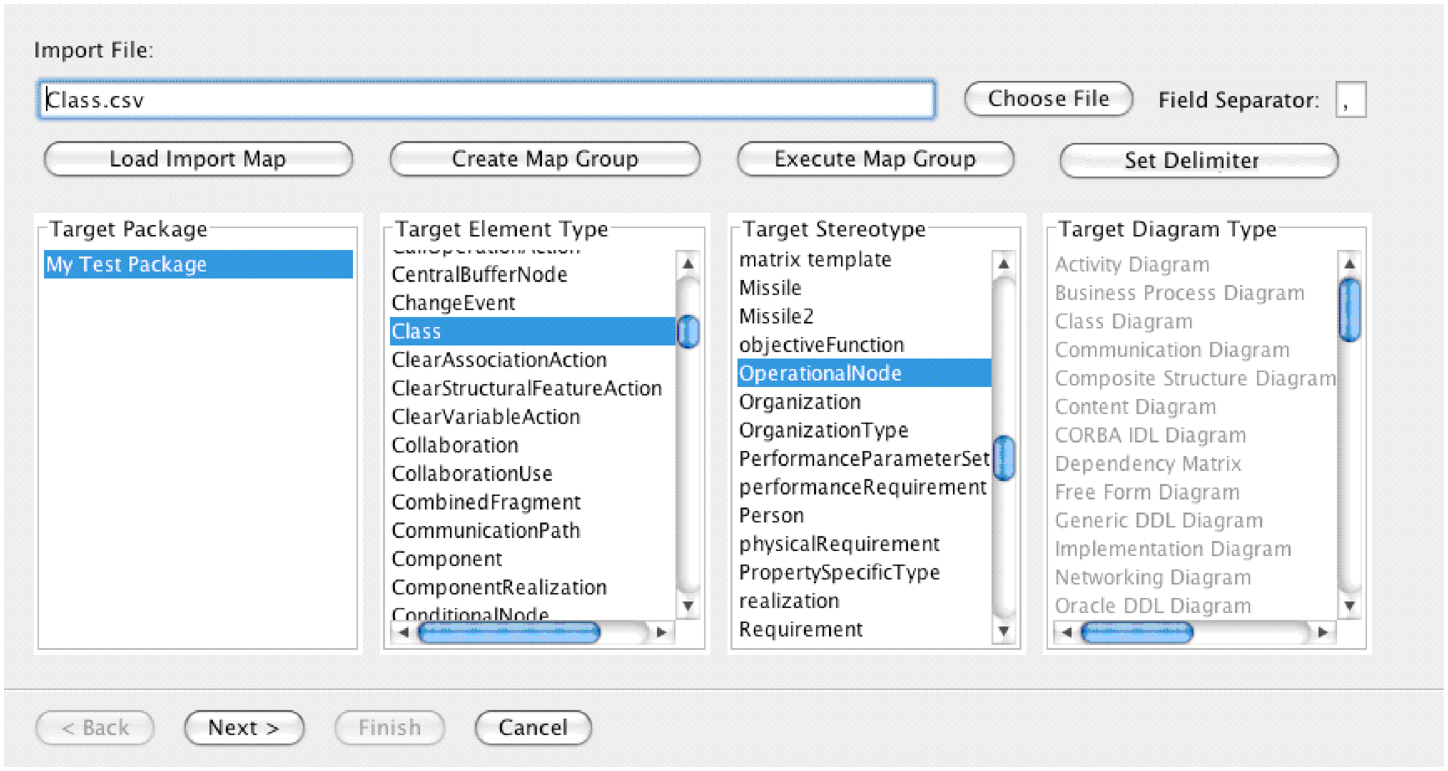
After pressing Finish, we see that several OV-2 diagrams have been created:


Contain..
Inheritance
Diagrams
<> Model Ex..
Containment
Data
DoDAF customization [DoDAF customization.m
MD Customization for SysML [MD_customizatio
My Test Package
Diag1
Diag2
Diag3
Diag4
UML Standard Profile [UML_Standard_Profile.xn
DoDAF Profile [DoDAF_Profile_2.0.mdzip]
SysML Profile [SysML Profile.mdzip]
UseCase Description Profile [UseCase_Profile.x
Diag1
Common
Note
Text Box
Anchor
Containment
Dependency
Image Shape
Separator
OV-2
Operational N...
Needline
Information E...
<<OV-2>>
OV-2 [Diag1]

Creating Operational Nodes

We can now populate them. This is even easier as it is just a matter of applying stereotypes. Let's create some operational nodes:

Bring up the plug-in and replicate the steps from Creating and Populating Class diagrams to create the class elements, but this time, choose the OperationalNode stereotype.


Import File: Class.csv

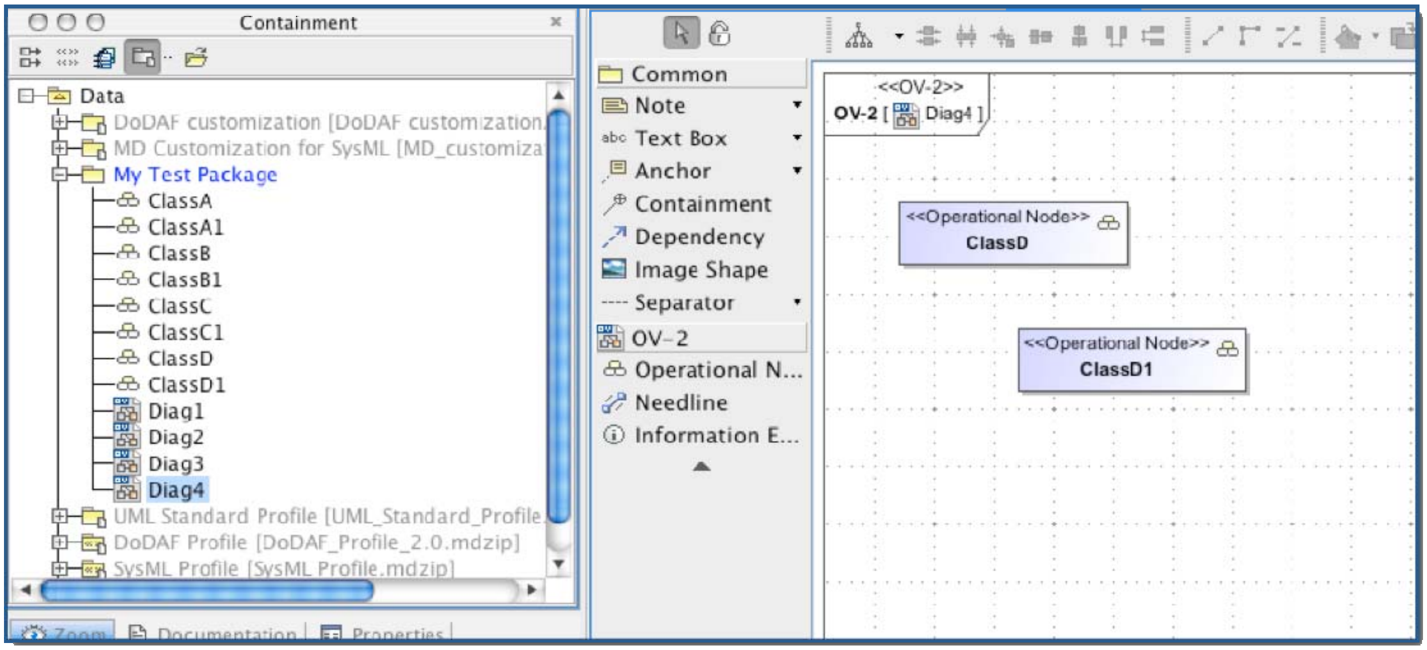
Field Separator: ,

Target Package: My Test Package

Target Element Type: Class

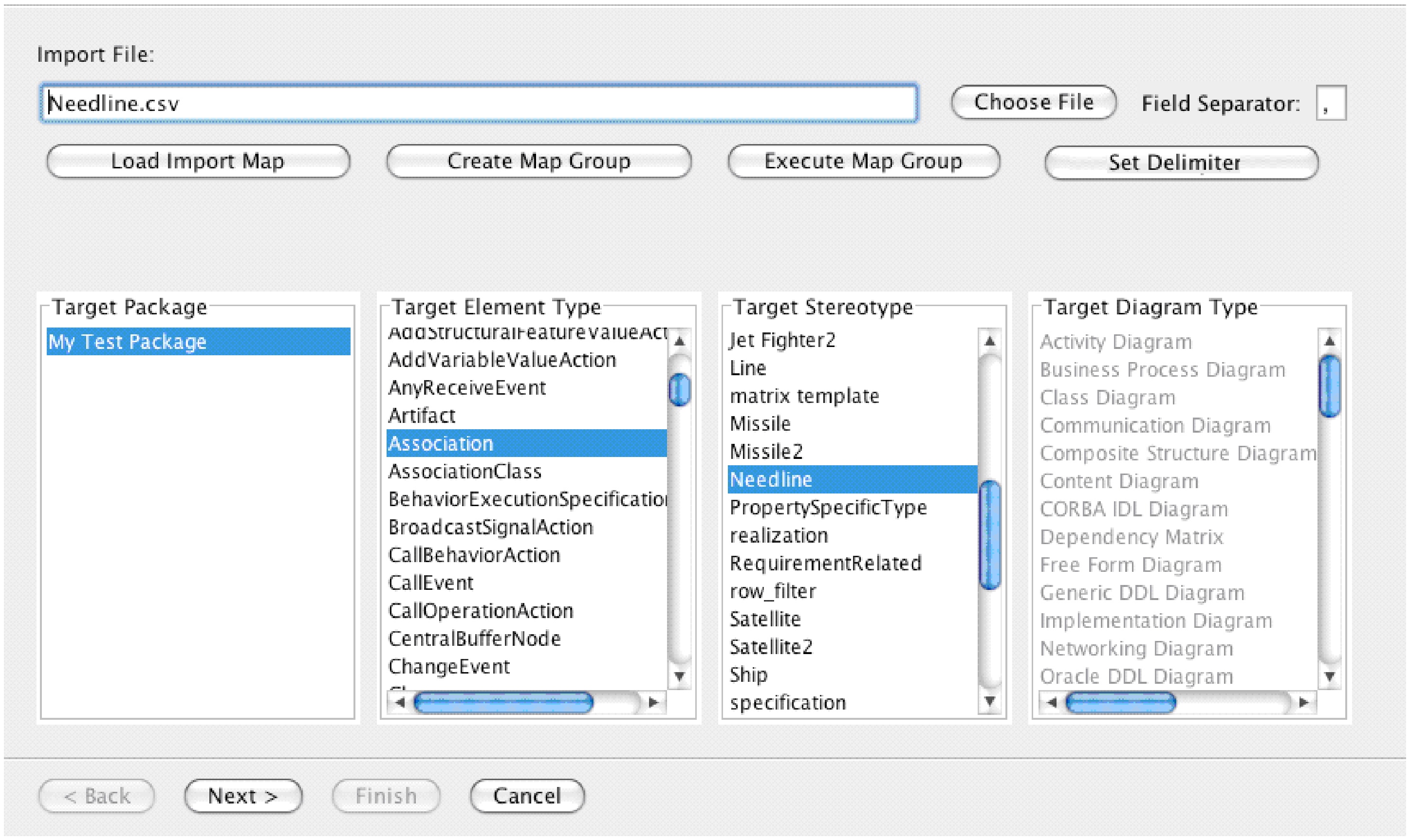
Target Stereotype: OperationalNode

When finished, the containment area and diagrams look like this:



Creating Needlines

Now, we'll draw the Needlines. Again just like creating the Class associations, only with a stereotype applied.



Import File: Needline.csv

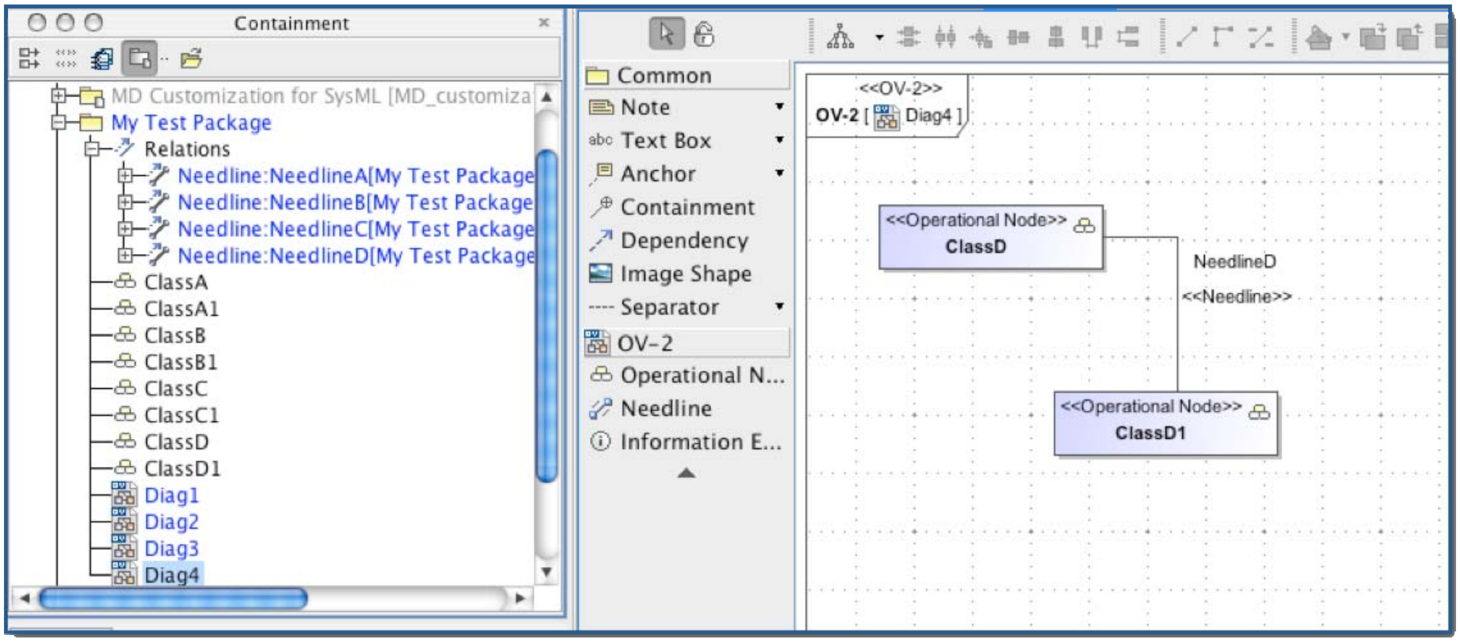
Field Separator: ,

Target Package: My Test Package

Target Element Type: Association

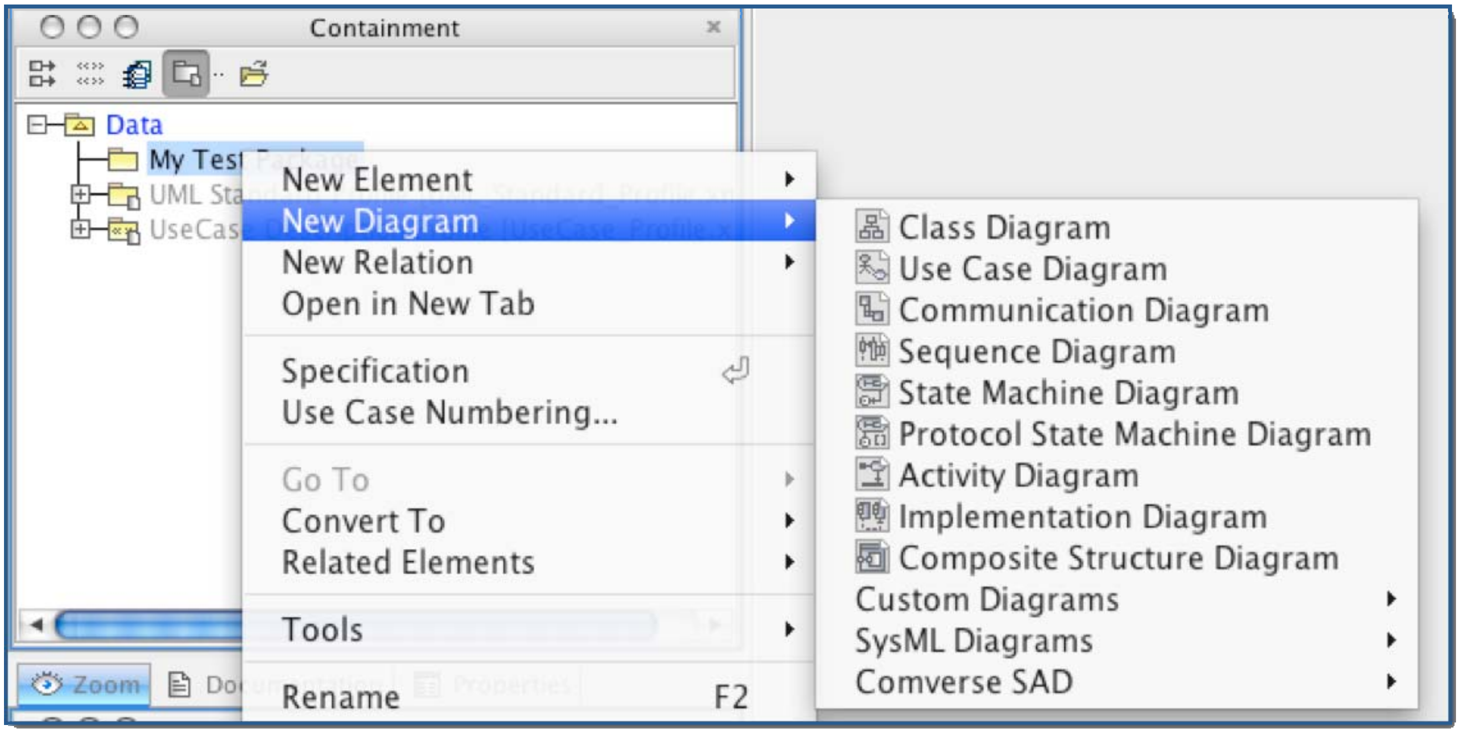
Target Stereotype: Needline

When done, the containment area and diagram will look like this:

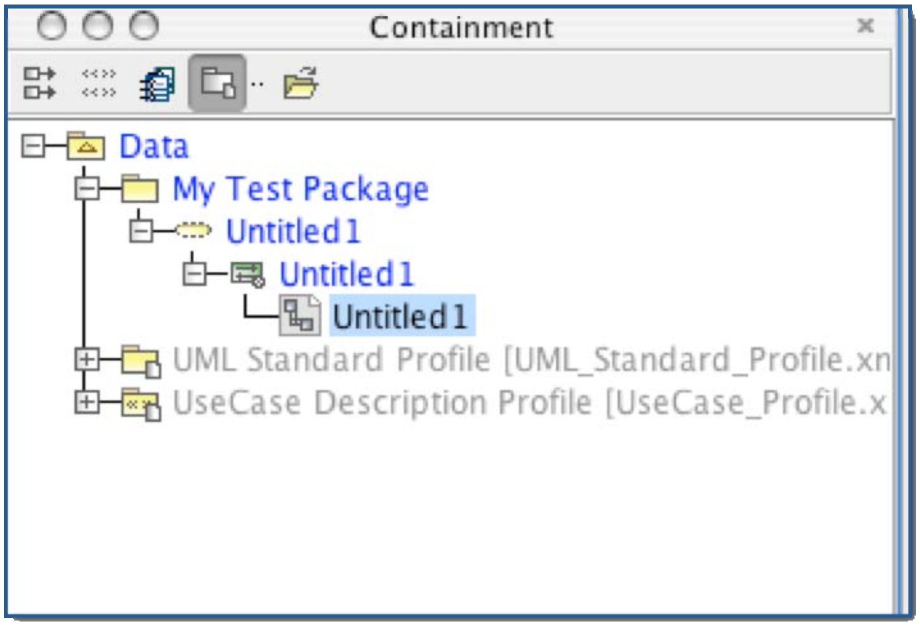


Prototyping Elements in a Test Model Before you Begin

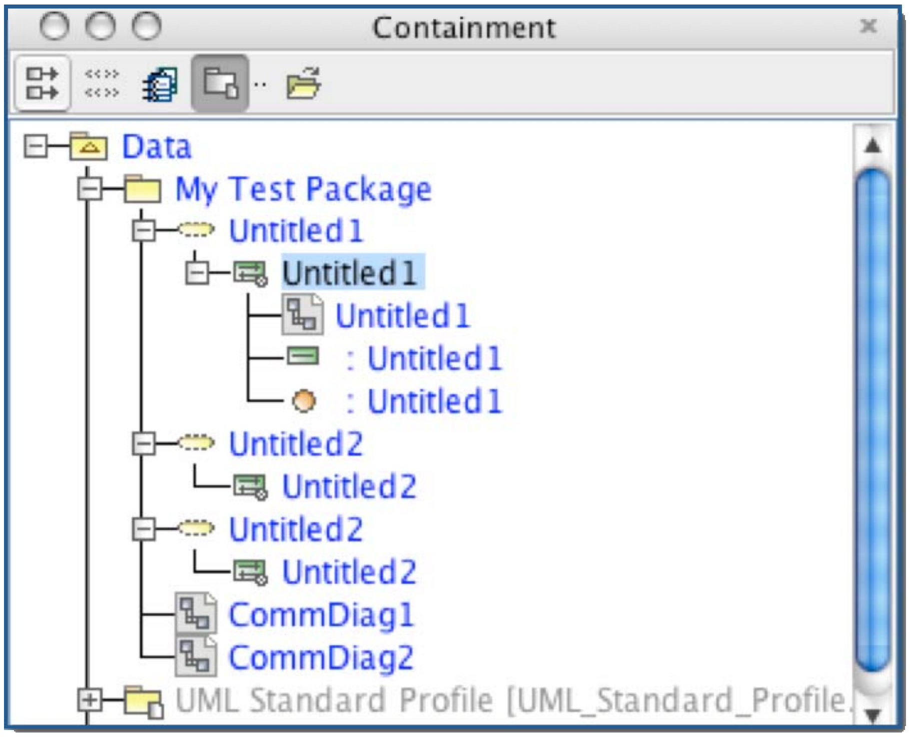
In order to avoid confusion, it pays to test out what you want to do in an empty model before you try the same thing using the plug-in. For example, create a new Communication Diagram:



Afterwards, your containment area will look like this:


Containment
Data
My Test Package
Untitled1
Untitled1
Untitled1
UML Standard Profile [UML_Standard_Profile.xn
UseCase Description Profile [UseCase_Profile.x

The element at the bottom of the tree is the Diagram. It is owned by an Interaction, which is in turn owned by a Collaboration, which is finally owned by the package. If you had simply used the plug-in to create the diagrams and let it assign ownership to the package, here is what your containment area would look like after you were done:

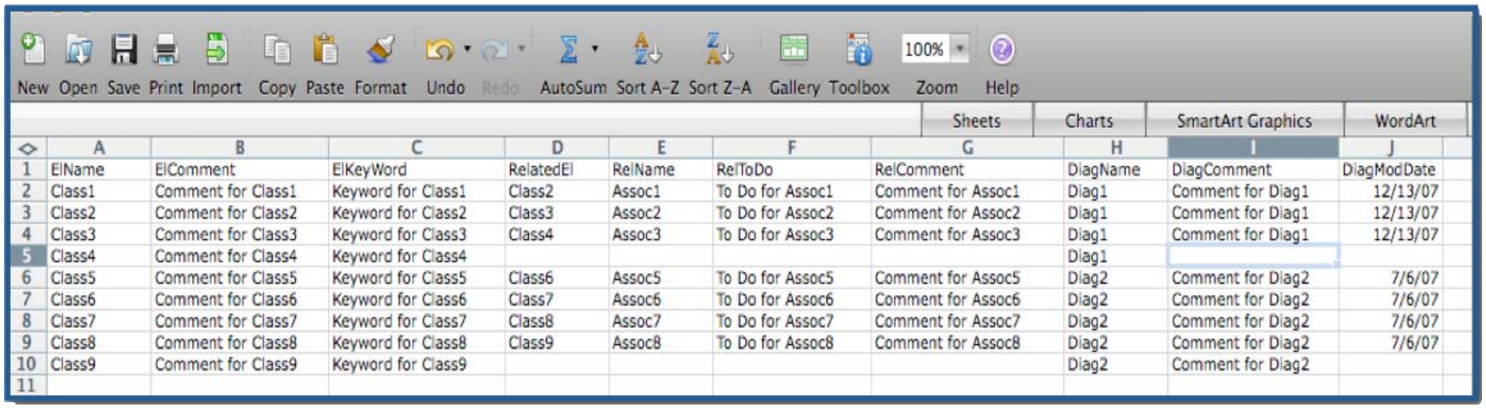


As you can see, the diagrams make it into the model, but since they’re not part of the structure that MagicDraw expects, they do not work properly. Attempts to add Lifelines to them fail.

Using one CSV File to Import Multiple Types of Data

Up to this point, the examples have all been of CSV files that contain very specific information. However, some databases may not contain that level of discreetness or it may be difficult to get the data in such discreet sets. This example shows how one would import different elements from the same set of data.

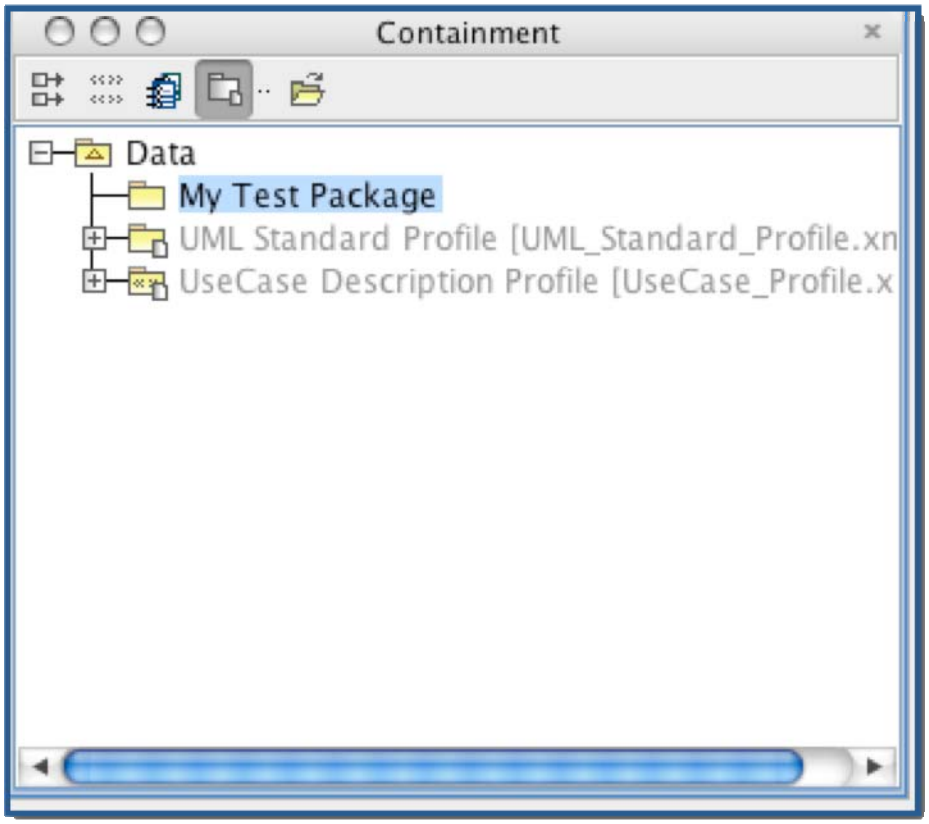
First, let's examine the data:



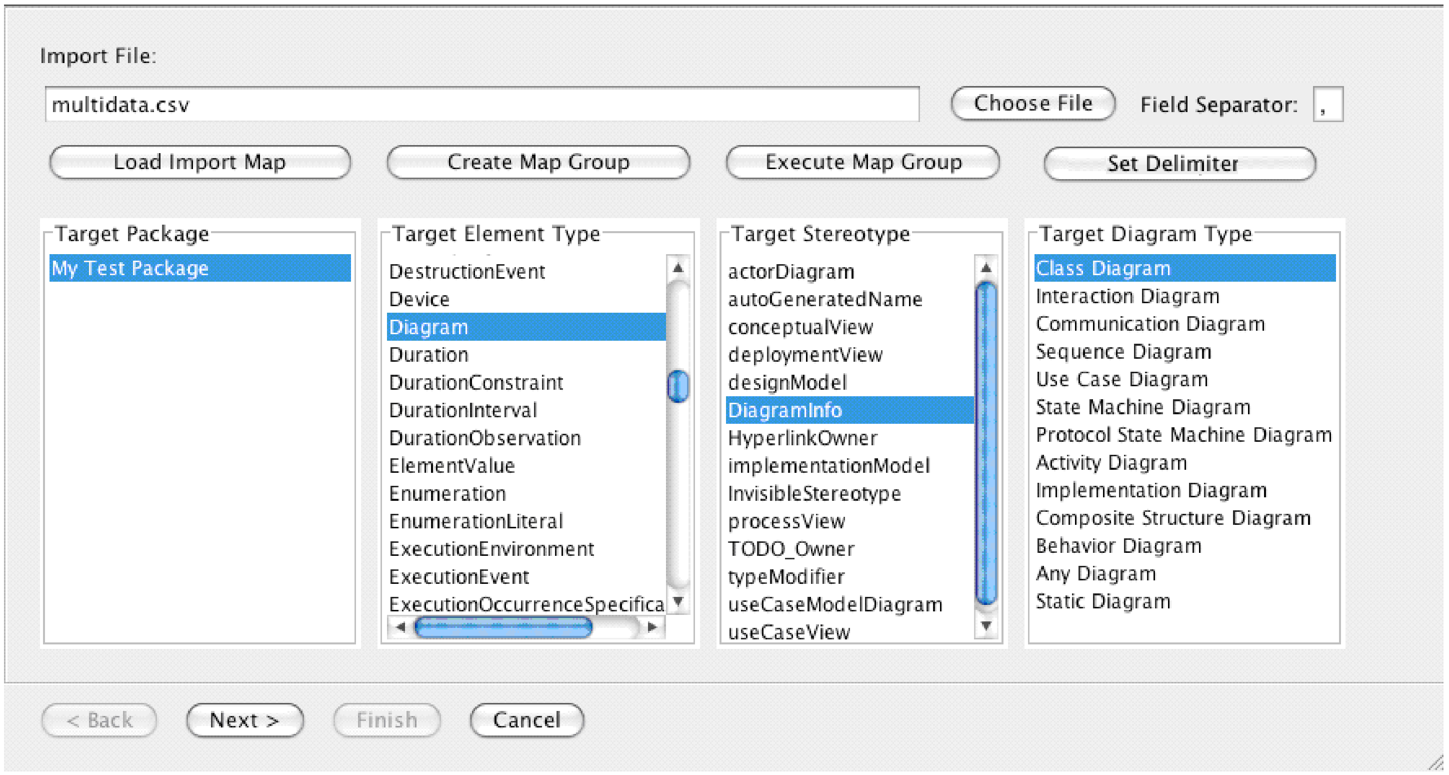
	A	B	C	D	E	F	G	H	I	J
1	ElName	ElComment	ElKeyWord	RelatedEl	RelName	RelToDo	RelComment	DiagName	DiagComment	DiagModDate
2	Class1	Comment for Class1	Keyword for Class1	Class2	Assoc1	To Do for Assoc1	Comment for Assoc1	Diag1	Comment for Diag1	12/13/07
3	Class2	Comment for Class2	Keyword for Class2	Class3	Assoc2	To Do for Assoc2	Comment for Assoc2	Diag1	Comment for Diag1	12/13/07
4	Class3	Comment for Class3	Keyword for Class3	Class4	Assoc3	To Do for Assoc3	Comment for Assoc3	Diag1	Comment for Diag1	12/13/07
5	Class4	Comment for Class4	Keyword for Class4					Diag1		
6	Class5	Comment for Class5	Keyword for Class5	Class6	Assoc5	To Do for Assoc5	Comment for Assoc5	Diag2	Comment for Diag2	7/6/07
7	Class6	Comment for Class6	Keyword for Class6	Class7	Assoc6	To Do for Assoc6	Comment for Assoc6	Diag2	Comment for Diag2	7/6/07
8	Class7	Comment for Class7	Keyword for Class7	Class8	Assoc7	To Do for Assoc7	Comment for Assoc7	Diag2	Comment for Diag2	7/6/07
9	Class8	Comment for Class8	Keyword for Class8	Class9	Assoc8	To Do for Assoc8	Comment for Assoc8	Diag2	Comment for Diag2	7/6/07
10	Class9	Comment for Class9	Keyword for Class9					Diag2	Comment for Diag2	
11										

Columns A, B, and C contain data about some Class elements we want to import. Columns D though G contain information about other classes that are associated with the class named in column A. Columns H, I, and J specify information about the diagrams that these classes and their relationships reside on. Notice how some of the data in rows 5 and 10 is blank. That's because there is no relationship to be created with the element in column A. If you were pulling this data out of a relational database, this sort of structure would be the result of an outer join.

Now, look at the containment area:


Containment
Data
My Test Package
UML Standard Profile [UML_Standard_Profile.xn
UseCase Description Profile [UseCase_Profile.x

Now, we bring up the plug-in to create the diagrams:



Import File: multidata.csv

Field Separator: ,

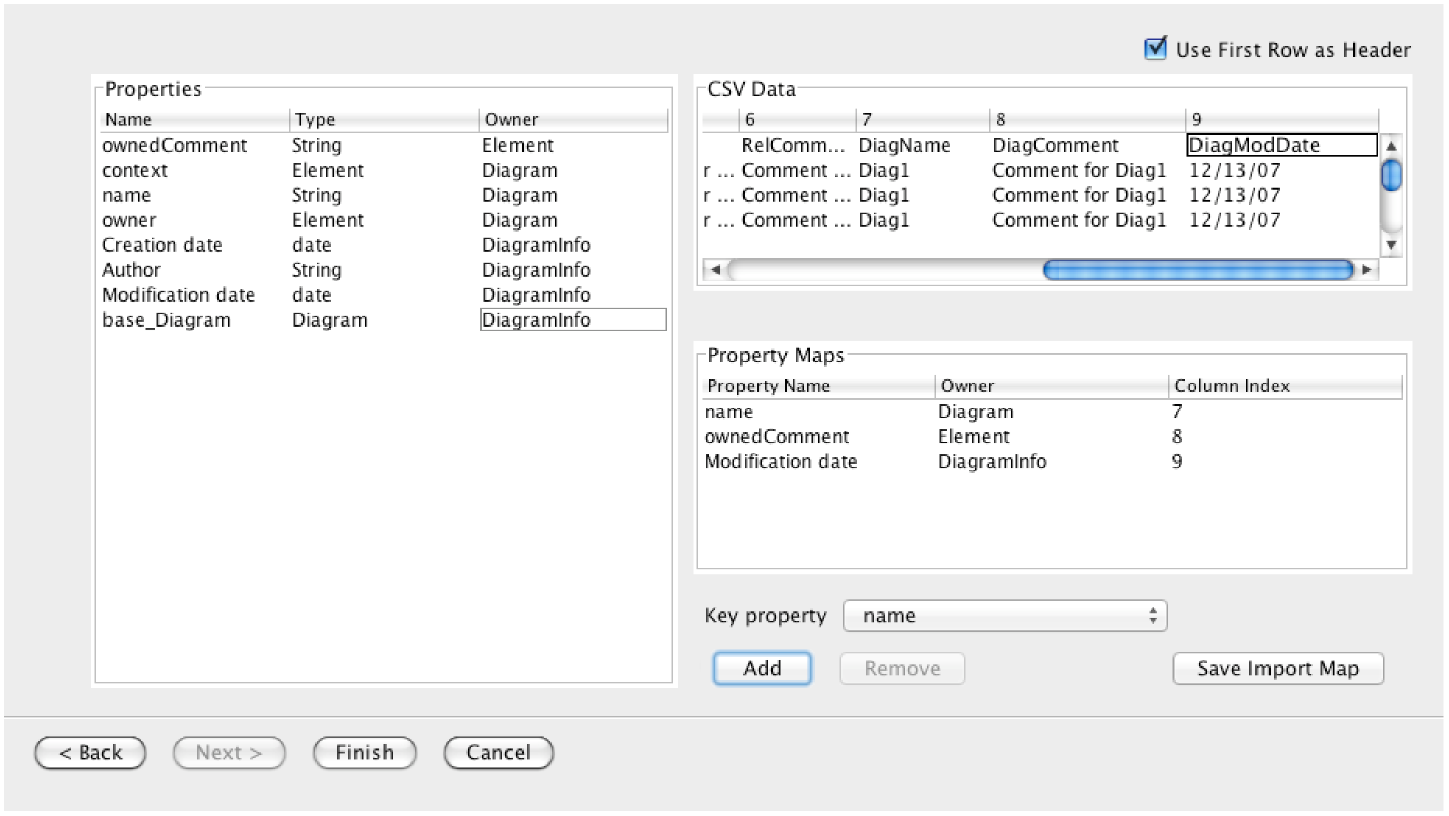
Target Package:
[✓ My Test Package]

Target Element Type:
DestructionEvent
Device
[✓ Diagram]
Duration
DurationConstraint
DurationInterval
DurationObservation
ElementValue
Enumeration
EnumerationLiteral
ExecutionEnvironment
ExecutionEvent
ExecutionOccurrenceSpecifica

Target Stereotype:
actorDiagram
autoGeneratedName
conceptualView
deploymentView
designModel
[✓ DiagramInfo]
HyperlinkOwner
implementationModel
InvisibleStereotype
processView
TODO_Owner
typeModifier
useCaseModelDiagram
useCaseView

Target Diagram Type:
[✓ Class Diagram]
Interaction Diagram
Communication Diagram
Sequence Diagram
Use Case Diagram
State Machine Diagram
Protocol State Machine Diagram
Activity Diagram
Implementation Diagram
Composite Structure Diagram
Behavior Diagram
Any Diagram
Static Diagram

Press Next to move to the next screen.



[✓] Use First Row as Header

Properties

Name	Type	Owner
ownedComment	String	Element
context	Element	Diagram
name	String	Diagram
owner	Element	Diagram
Creation date	date	DiagramInfo
Author	String	DiagramInfo
Modification date	date	DiagramInfo
base_Diagram	Diagram	DiagramInfo

CSV Data

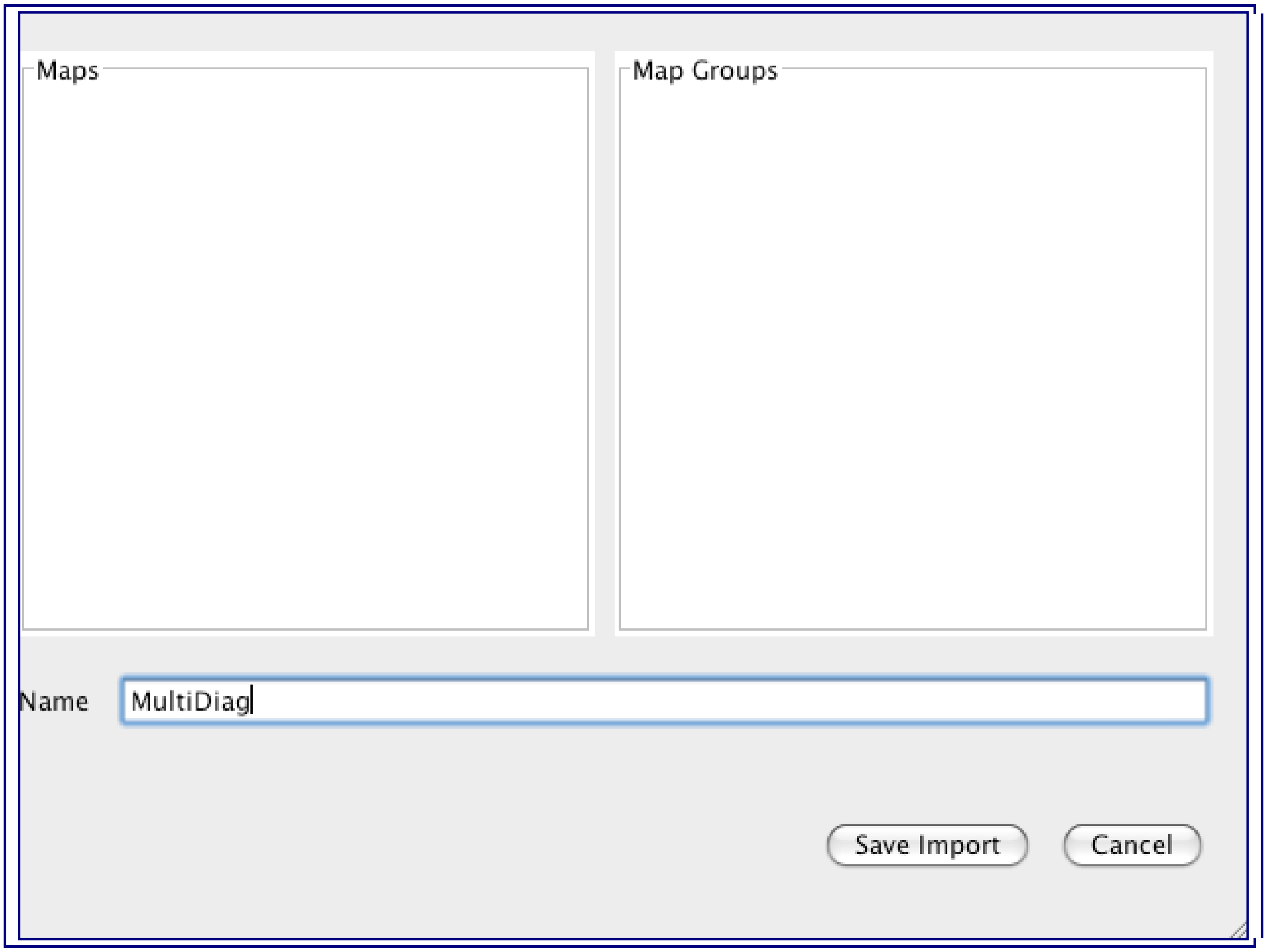
	6	7	8	9
	RelComm...	DiagName	DiagComment	DiagModDate
r ...	Comment ...	Diag1	Comment for Diag1	12/13/07
r ...	Comment ...	Diag1	Comment for Diag1	12/13/07
r ...	Comment ...	Diag1	Comment for Diag1	12/13/07

Property Maps

Property Name	Owner	Column Index
name	Diagram	7
ownedComment	Element	8
Modification date	DiagramInfo	9

Key property: name

Notice how we've only selected the columns that are concerned with the diagrams. Now, we'll save this import to the model. Press Save Import Map.



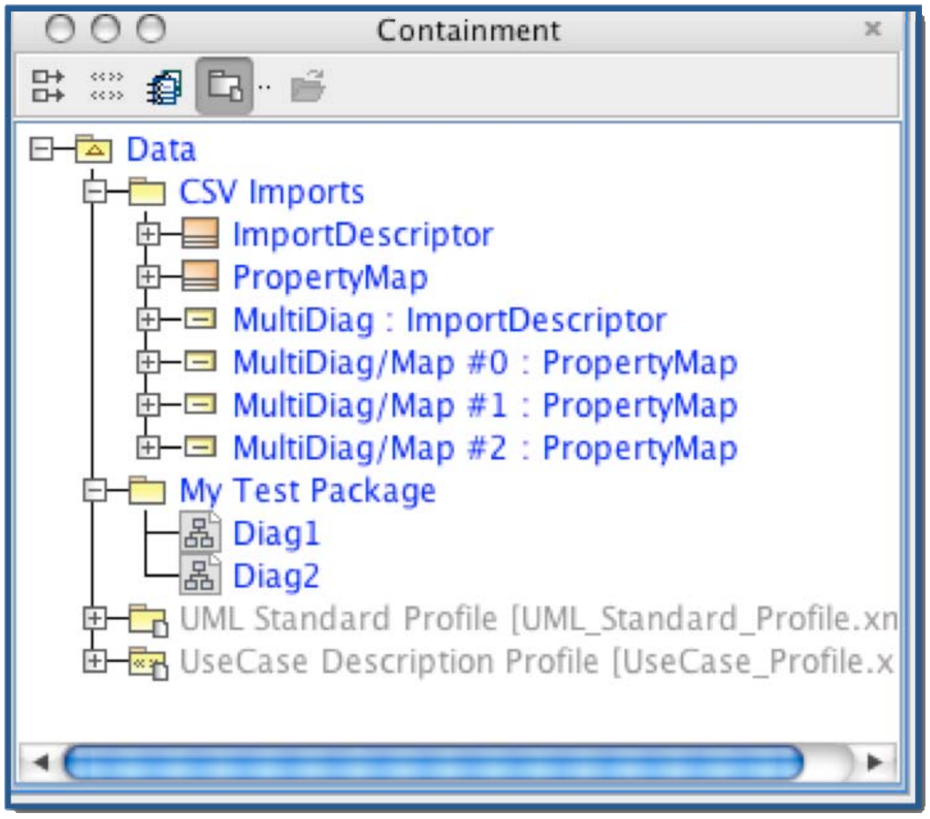
Press Save Import to save and go back to the Mappings Screen.

When you press finish, you'll see the following text in the Messages Window:

```
Import Wizard Started
Import Started
Row 1 loaded: name = Diag1 Element ownedComment = Comment for
Diag1 DiagramInfo Modification date = 12/13/07
Row 2 loaded: name = Diag1 Element ownedComment = Comment for
Diag1 DiagramInfo Modification date = 12/13/07
Row 3 loaded: name = Diag1 Element ownedComment = Comment for
Diag1 DiagramInfo Modification date = 12/13/07
Error on row: 4 No value in name column. Skipping this row.
```

```
Row 5 loaded: name = Diag2 Element ownedComment = Comment for
Diag2 DiagramInfo Modification date = 7/6/07
Row 6 loaded: name = Diag2 Element ownedComment = Comment for
Diag2 DiagramInfo Modification date = 7/6/07
Row 7 loaded: name = Diag2 Element ownedComment = Comment for
Diag2 DiagramInfo Modification date = 7/6/07
Row 8 loaded: name = Diag2 Element ownedComment = Comment for
Diag2 DiagramInfo Modification date = 7/6/07
Error on row: 9 No value in name column. Skipping this row.
Import Complete - 9 records processed.
```

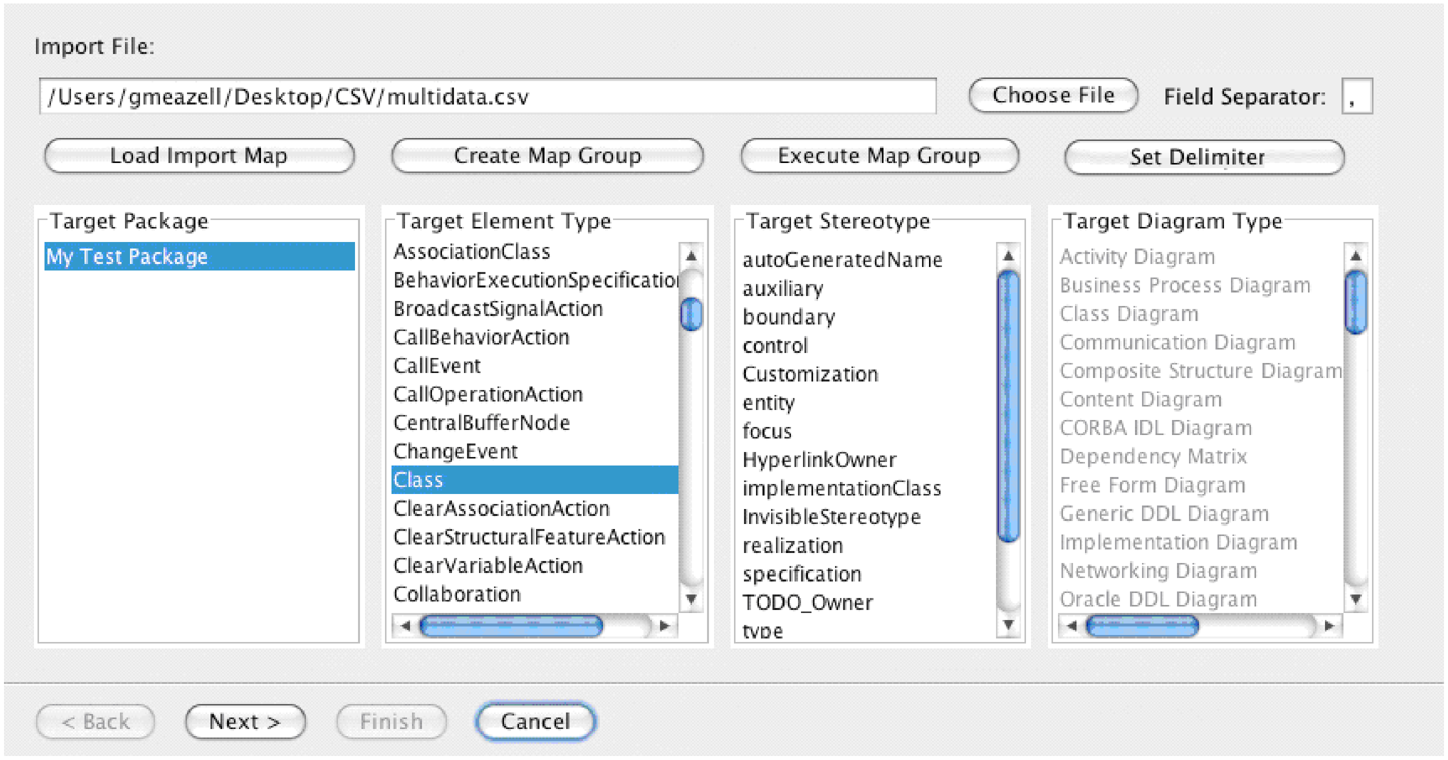
We will also see changes to the containment area:



```
Data
  CSV Imports
    ImportDescriptor
    PropertyMap
    MultiDiag : ImportDescriptor
    MultiDiag/Map #0 : PropertyMap
    MultiDiag/Map #1 : PropertyMap
    MultiDiag/Map #2 : PropertyMap
  My Test Package
    Diag1
    Diag2
  UML Standard Profile [UML_Standard_Profile.xn
  UseCase Description Profile [UseCase_Profile.x
```

The CSV Imports package was created and populated and the My Test Package was populated with the two diagrams. Notice how the plug-in processed seven valid lines, but only two diagrams were created. This is because the diagram information was duplicated. Loading duplicate data does not cause errors, it merely overwrites the previous data.

Now, let's move on to importing the classes from the file. Bring up the plug-in.



Import File: /Users/gmeazell/Desktop/CSV/multidata.csv

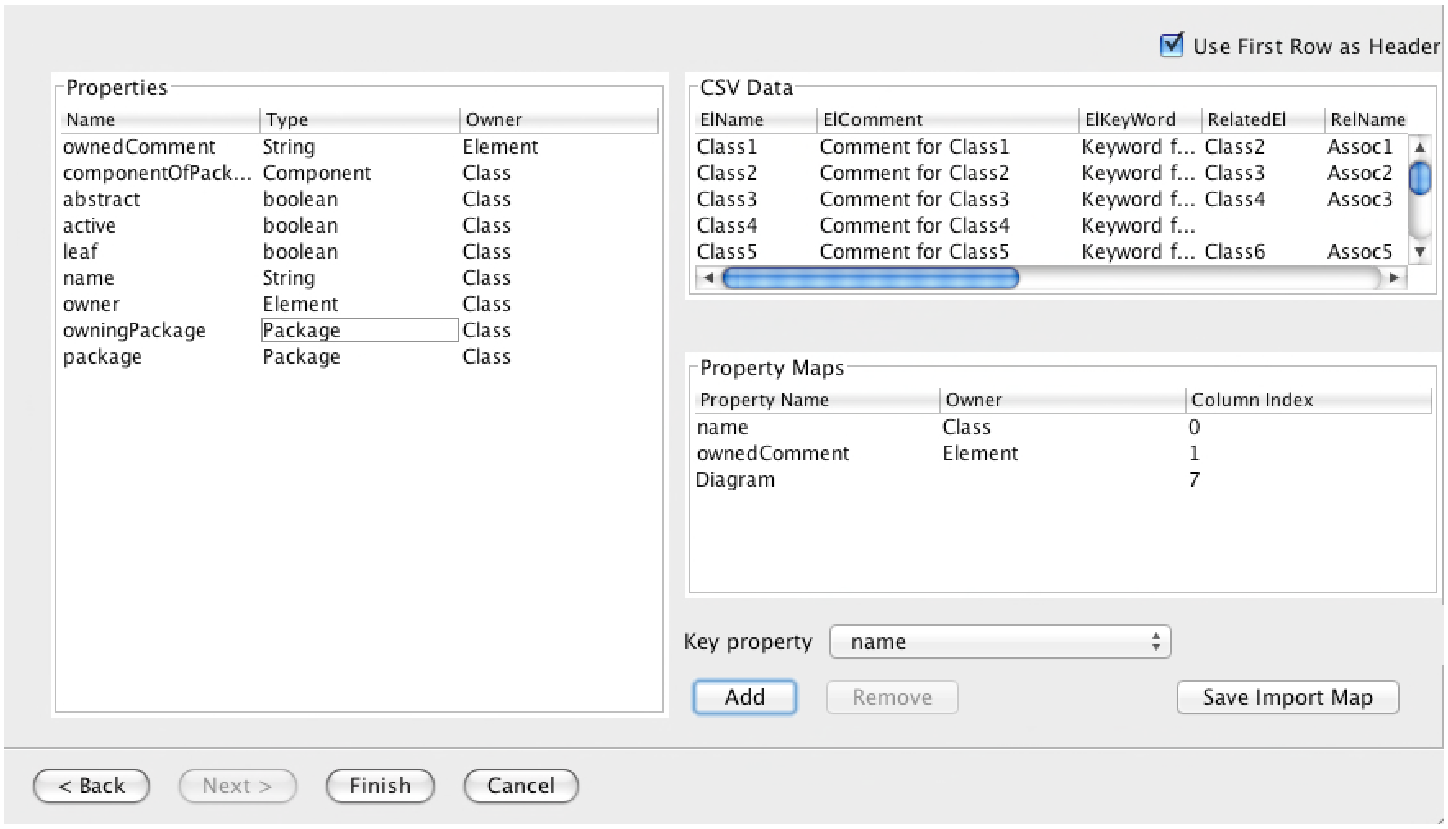
Field Separator: ,

Target Package: My Test Package

Target Element Type: Class

Target Stereotype:
autoGeneratedName
auxiliary
boundary
control
Customization
entity
focus
HyperlinkOwner
implementationClass
InvisibleStereotype
realization
specification
TODO_Owner
type

Press Next to move to the next screen.



Use First Row as Header ✓

Properties

Name	Type	Owner
ownedComment	String	Element
componentOfPack...	Component	Class
abstract	boolean	Class
active	boolean	Class
leaf	boolean	Class
name	String	Class
owner	Element	Class
owningPackage	Package	Class
package	Package	Class

CSV Data

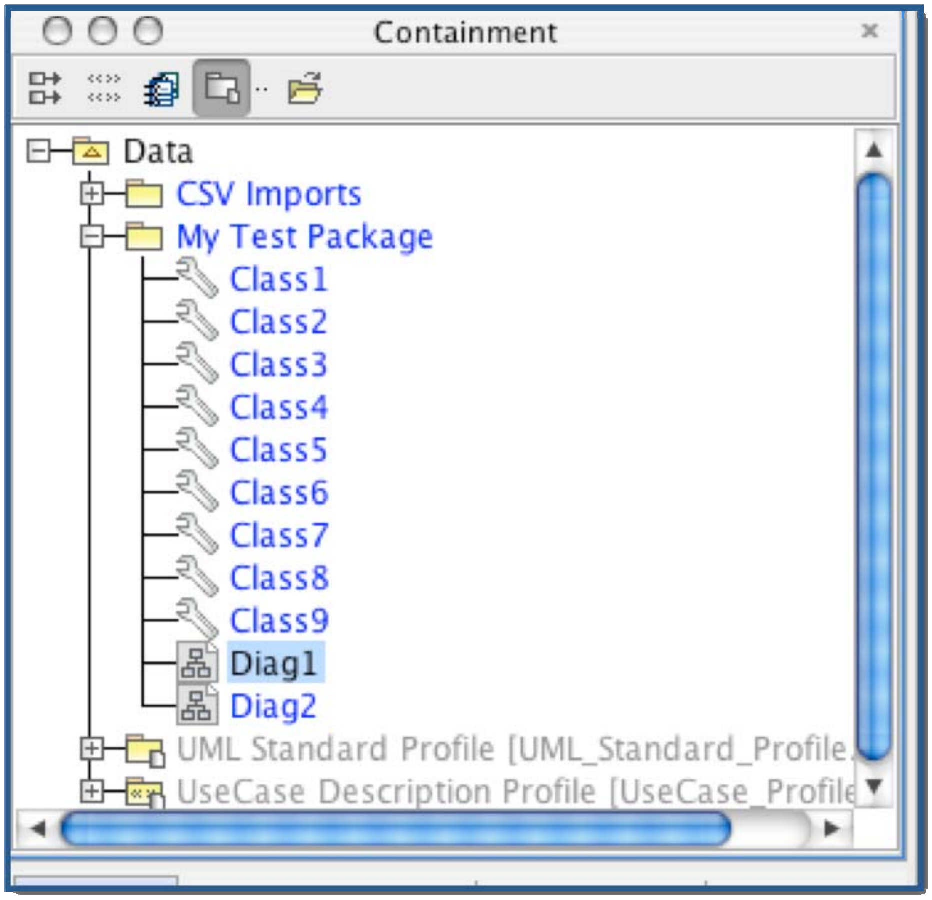
ElName	ElComment	ElKeyWord	RelatedEl	RelName
Class1	Comment for Class1	Keyword f...	Class2	Assoc1
Class2	Comment for Class2	Keyword f...	Class3	Assoc2
Class3	Comment for Class3	Keyword f...	Class4	Assoc3
Class4	Comment for Class4	Keyword f...		
Class5	Comment for Class5	Keyword f...	Class6	Assoc5

Property Maps

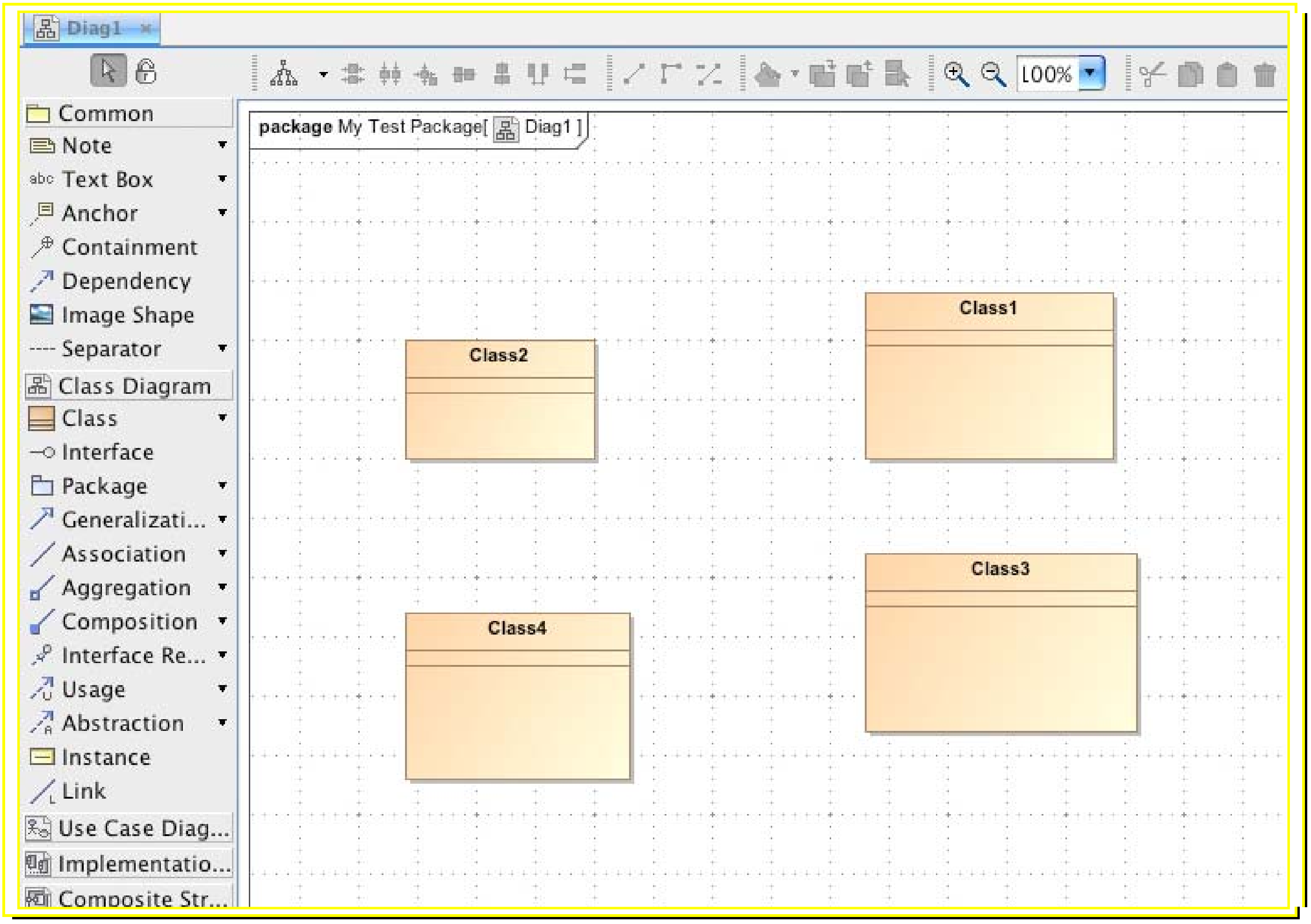
Property Name	Owner	Column Index
name	Class	0
ownedComment	Element	1
Diagram		7

Key property: name

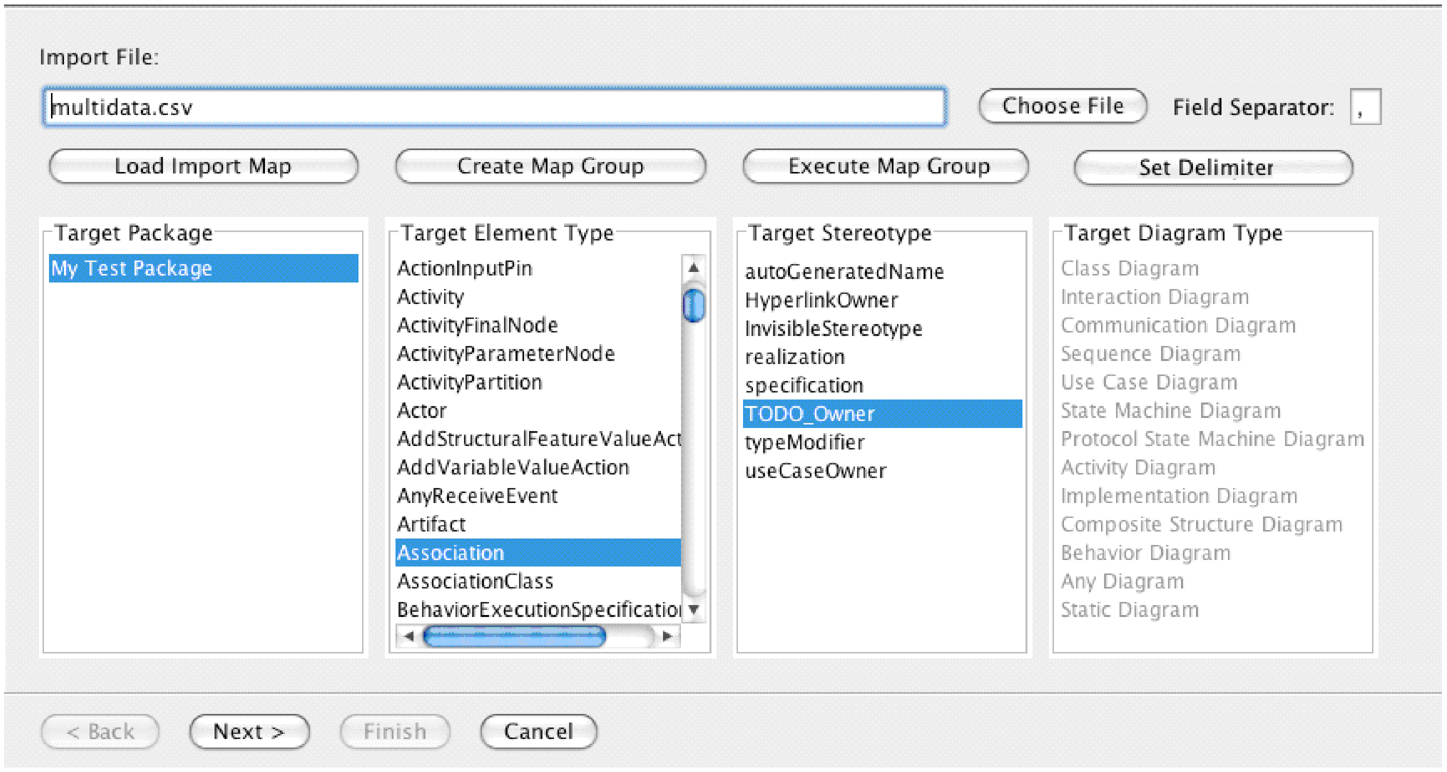
Again, we select only a subset of the columns. Press Finish and the containment area shows the created classes:



Inspection of diagram Diag1 reveals these changes:



Now, let's import the relationships. Bring up the plug-in again.


Import File: multidata.csv

Field Separator: ,

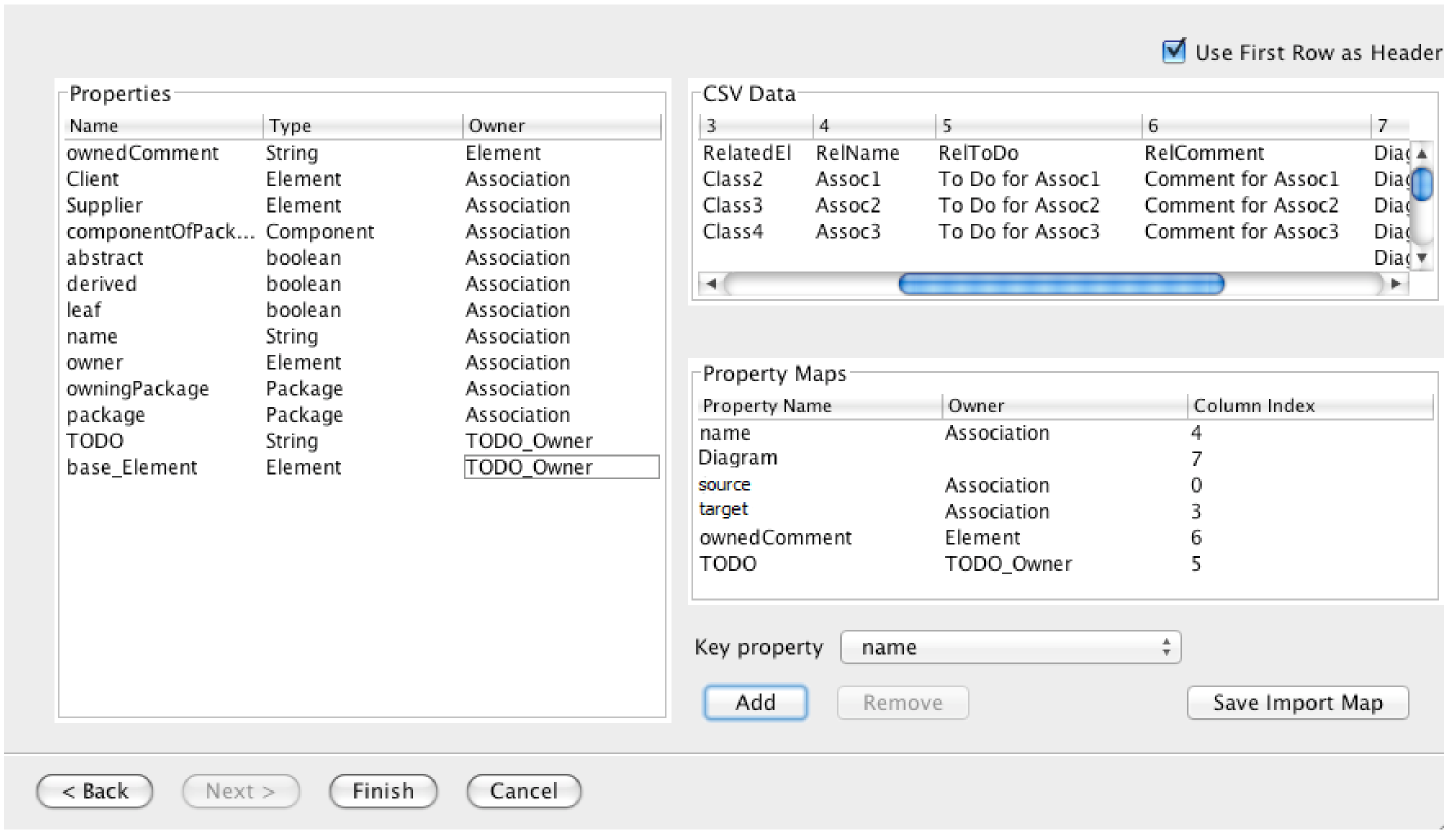
Target Package: My Test Package

Target Element Type: ActionInputPin, Activity, ActivityFinalNode, ActivityParameterNode, ActivityPartition, Actor, AddStructuralFeatureValueAct, AddVariableValueAction, AnyReceiveEvent, Artifact, Association, AssociationClass, BehaviorExecutionSpecificatio

Target Stereotype: autoGeneratedName, HyperlinkOwner, InvisibleStereotype, realization, specification, TODO_Owner, typeModifier, useCaseOwner

Target Diagram Type: Class Diagram, Interaction Diagram, Communication Diagram, Sequence Diagram, Use Case Diagram, State Machine Diagram, Protocol State Machine Diagram, Activity Diagram, Implementation Diagram, Composite Structure Diagram, Behavior Diagram, Any Diagram, Static Diagram

Press Next to move to the next screen.


Use First Row as Header

Properties

Name	Type	Owner
ownedComment	String	Element
Client	Element	Association
Supplier	Element	Association
componentOfPack...	Component	Association
abstract	boolean	Association
derived	boolean	Association
leaf	boolean	Association
name	String	Association
owner	Element	Association
owningPackage	Package	Association
package	Package	Association
TODO	String	TODO_Owner
base_Element	Element	TODO_Owner

CSV Data

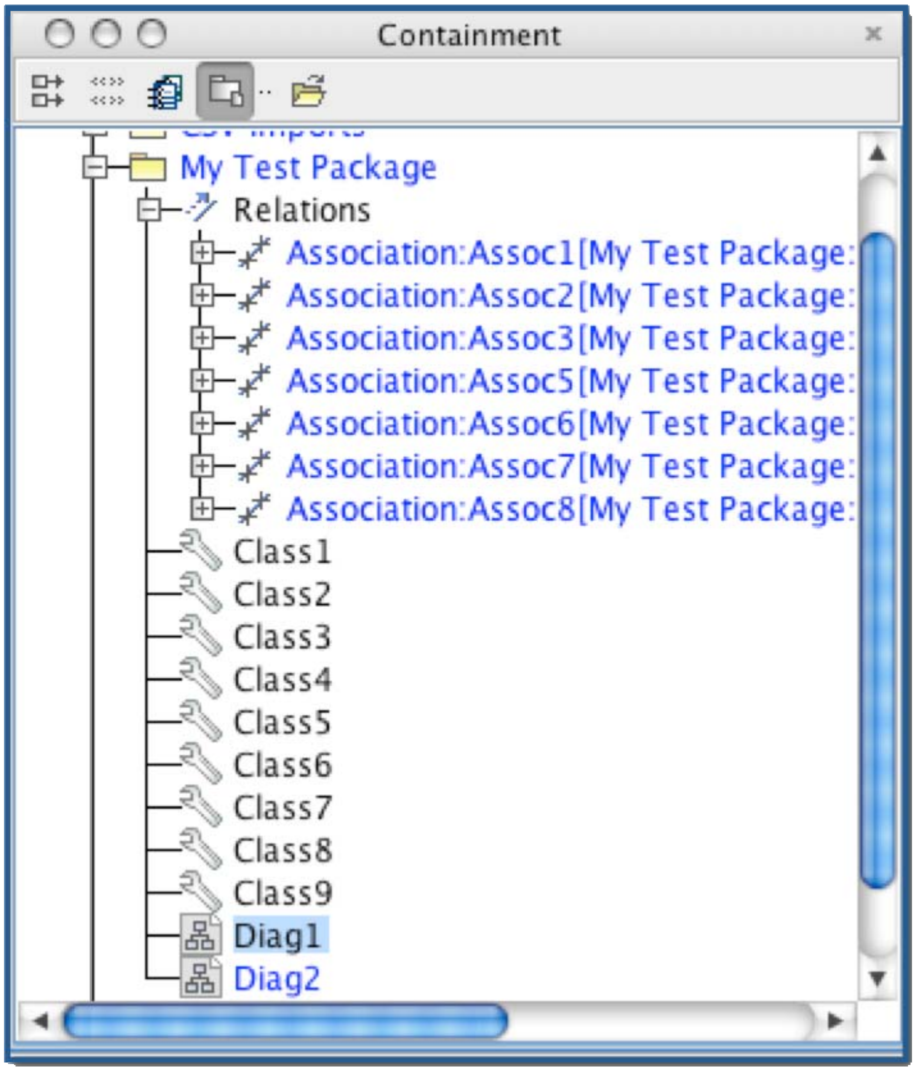
3	4	5	6	7
RelatedEl	RelName	RelToDo	RelComment	Dia
Class2	Assoc1	To Do for Assoc1	Comment for Assoc1	Dia
Class3	Assoc2	To Do for Assoc2	Comment for Assoc2	Dia
Class4	Assoc3	To Do for Assoc3	Comment for Assoc3	Dia
				Dia

Property Maps

Property Name	Owner	Column Index
name	Association	4
Diagram		7
source	Association	0
target	Association	3
ownedComment	Element	6
TODO	TODO_Owner	5

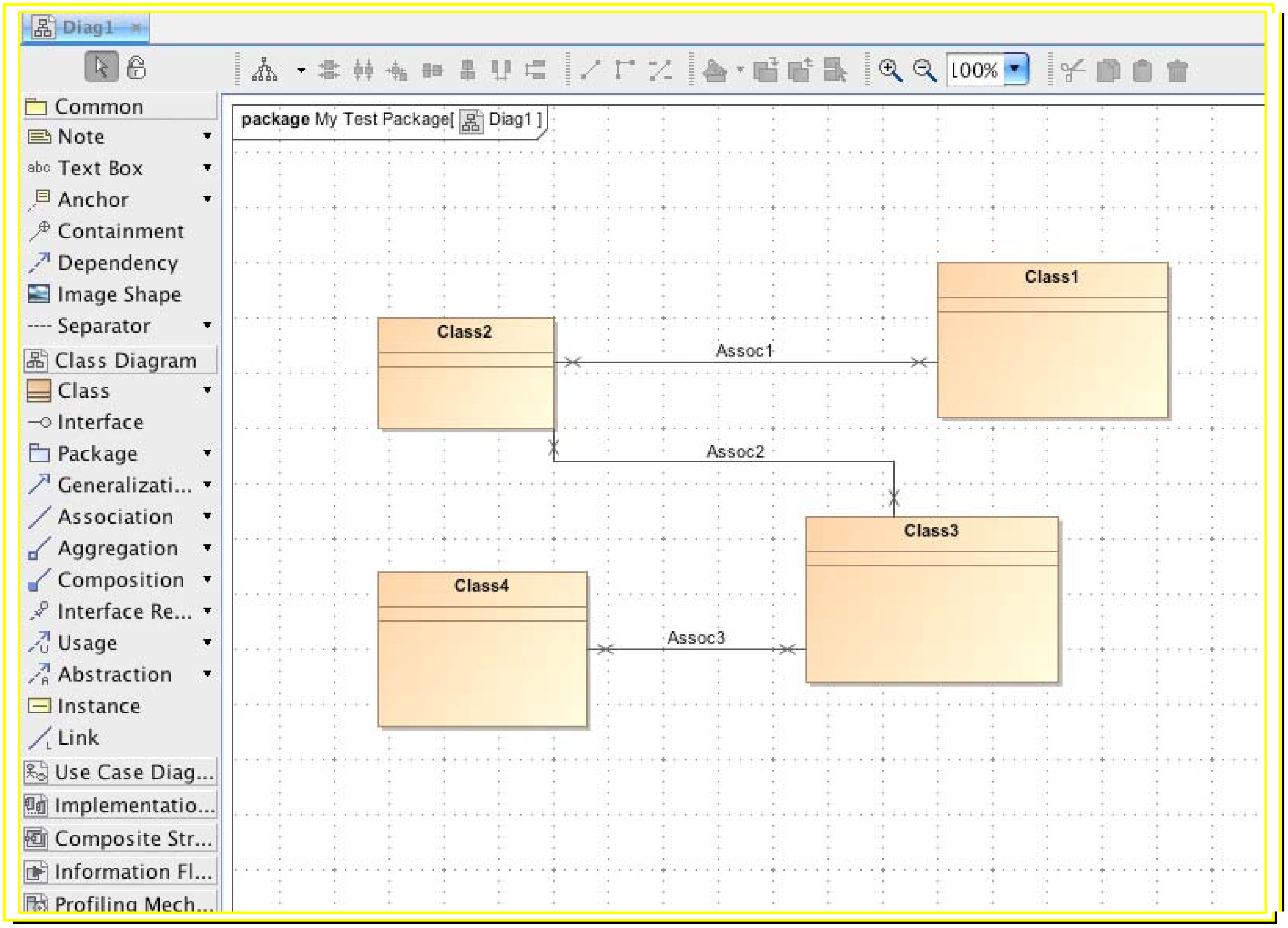
Key property: name

After pressing Finish, when we review the containment area, we see that the relationships have been added:



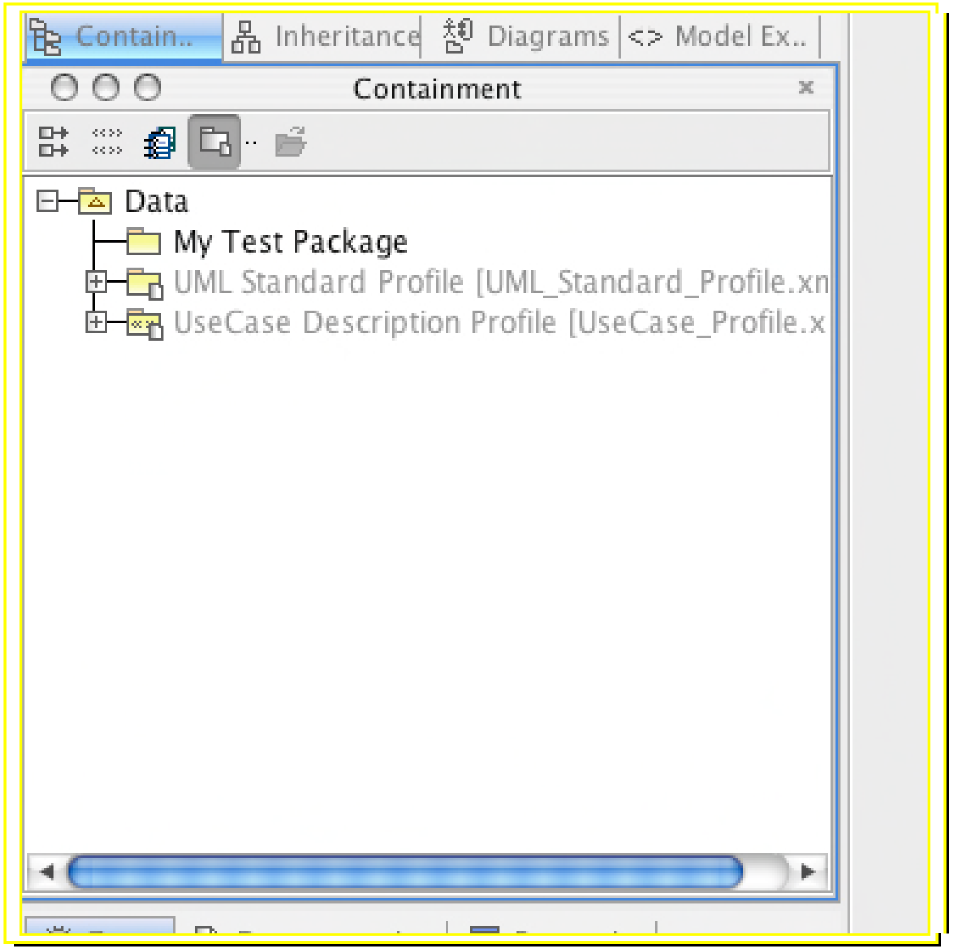
My Test Package
Relations
Association:Assoc1[My Test Package:
Association:Assoc2[My Test Package:
Association:Assoc3[My Test Package:
Association:Assoc5[My Test Package:
Association:Assoc6[My Test Package:
Association:Assoc7[My Test Package:
Association:Assoc8[My Test Package:
Class1
Class2
Class3
Class4
Class5
Class6
Class7
Class8
Class9
Diag1
Diag2

We can also see them on diagram Diag1:



Custom Model Elements Revisited: Requirements

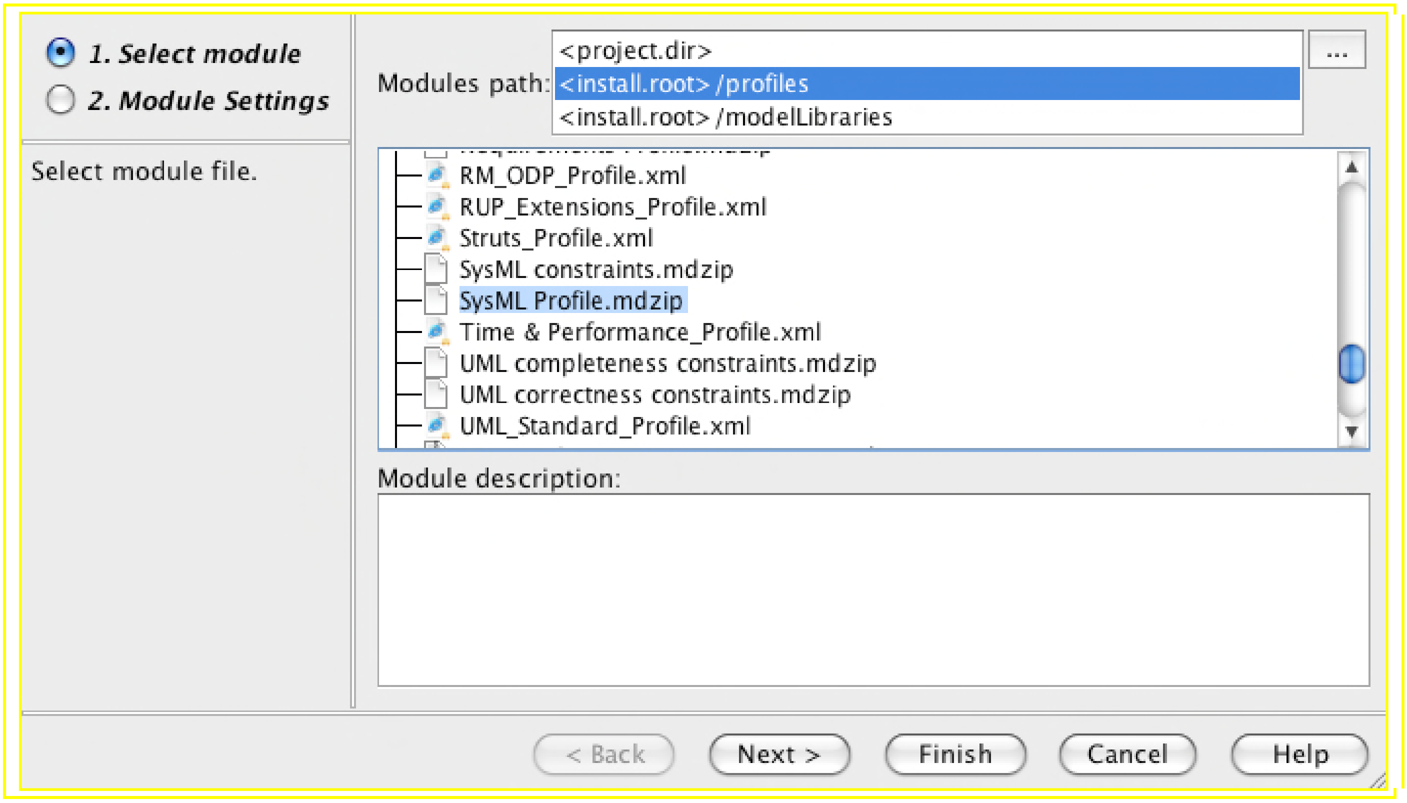
One of the more powerful aspects of MagicDraw is the ability to apply stereotypes to an element and thus affect the way the element is displayed and operates within your model. To further demonstrate this, we will build a SysML requirements diagram. First, we start with a new model.



Contain..
Inheritance
Diagrams
<> Model Ex..
Containment
Data
My Test Package
UML Standard Profile [UML_Standard_Profile.xn
UseCase Description Profile [UseCase_Profile.x

In this example, I already know that most of my model elements will not be owned by the package. Therefore, I can change the order of the import from the standard and import the elements first.

Start by loading the SysML profile module:



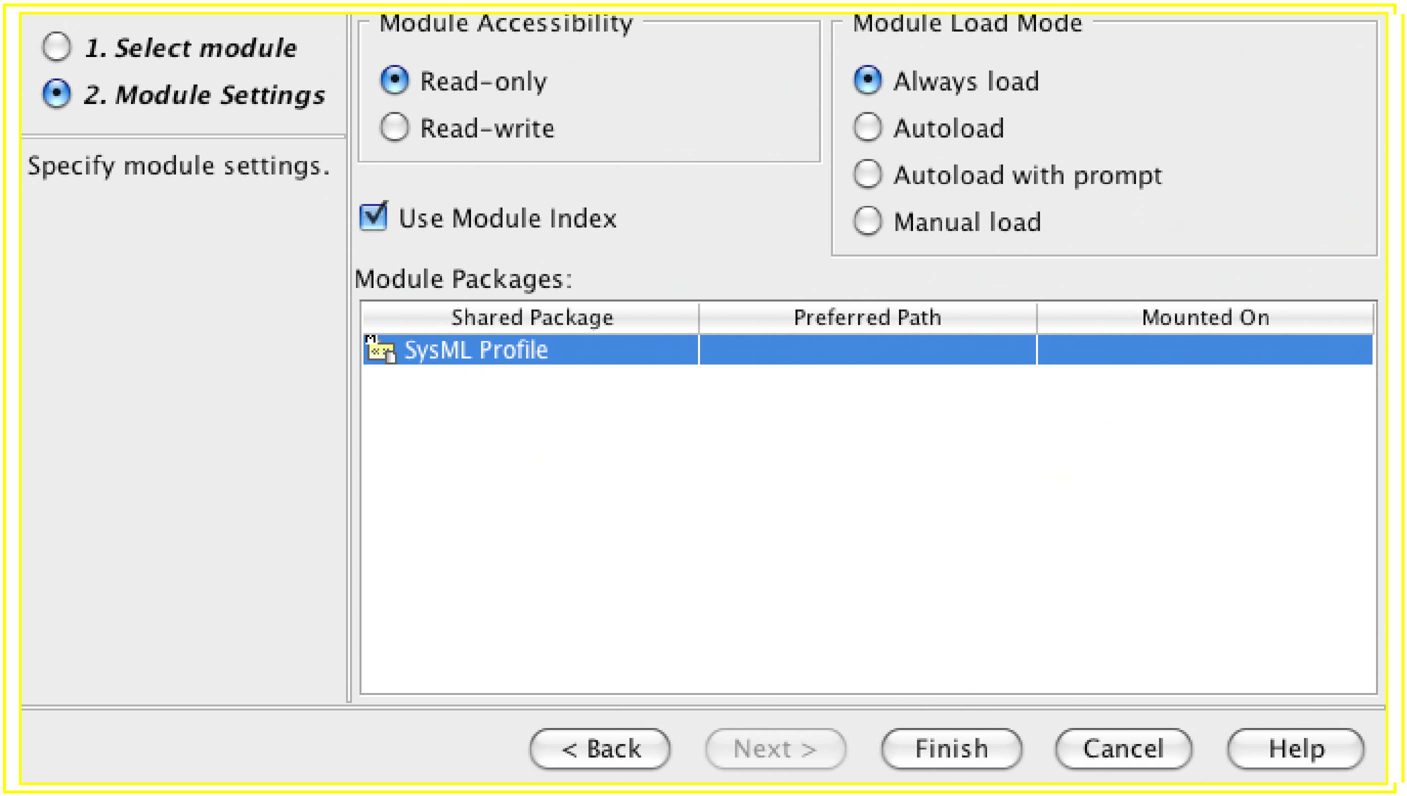
Select module file.

Modules path:
- <project.dir>
- <install.root>/profiles
- <install.root>/modelLibraries

- RM_ODP_Profile.xml
- RUP_Extensions_Profile.xml
- Struts_Profile.xml
- SysML constraints.mdzip
- SysML Profile.mdzip
- Time & Performance_Profile.xml
- UML completeness constraints.mdzip
- UML correctness constraints.mdzip
- UML_Standard_Profile.xml

Module description:

Press Next to move to the next screen.



Specify module settings.

Module Accessibility: [✓ Read-only] Read-write

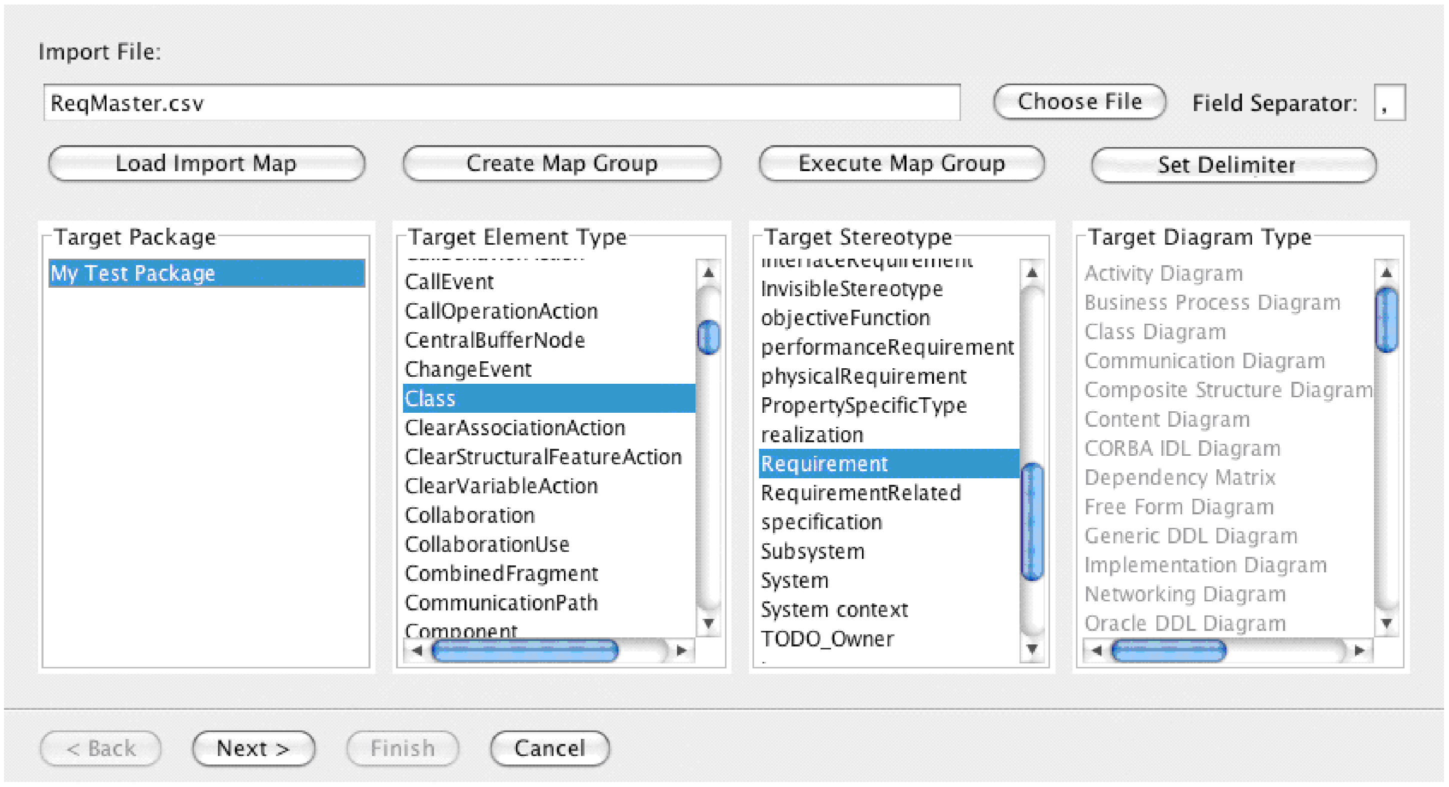
Module Load Mode: [✓ Always load] Autoload Autoload with prompt Manual load

[✓] Use Module Index

Module Packages:

Shared Package	Preferred Path	Mounted On
SysML Profile		

Press Finish to load the module. Now, start the plug-in.



Import File: ReqMaster.csv

Field Separator: ,

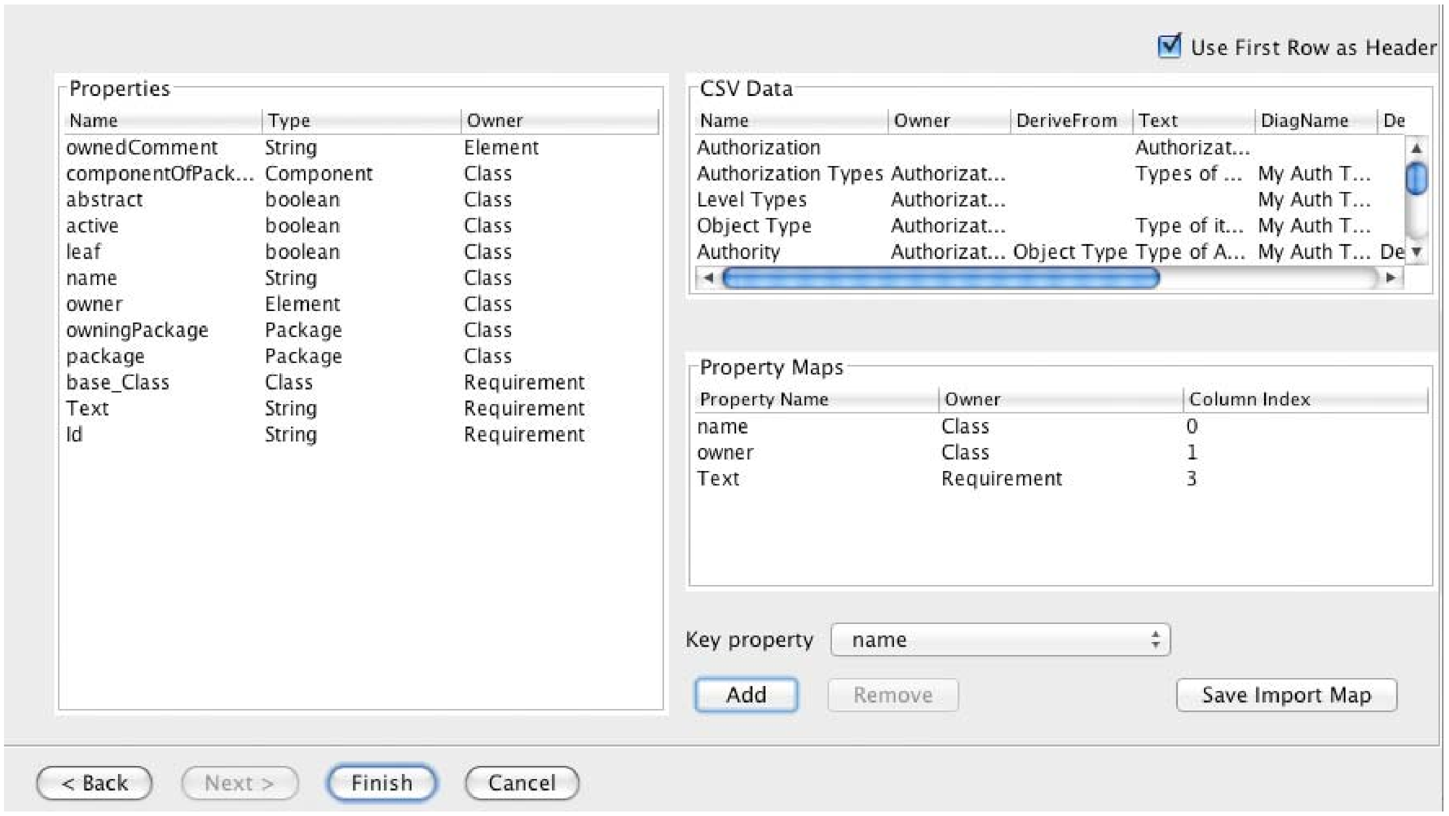
Target Package:
- My Test Package

Target Element Type:
- CallEvent
- CallOperationAction
- CentralBufferNode
- ChangeEvent
- Class
- ClearAssociationAction
- ClearStructuralFeatureAction
- ClearVariableAction
- Collaboration
- CollaborationUse
- CombinedFragment
- CommunicationPath
- Component

Target Stereotype:
- InvisibleStereotype
- objectiveFunction
- performanceRequirement
- physicalRequirement
- PropertySpecificType
- realization
- Requirement
- RequirementRelated
- specification
- Subsystem
- System
- System context
- TODO_Owner

Target Diagram Type:
- Activity Diagram
- Business Process Diagram
- Class Diagram
- Communication Diagram
- Composite Structure Diagram
- Content Diagram
- CORBA IDL Diagram
- Dependency Matrix
- Free Form Diagram
- Generic DDL Diagram
- Implementation Diagram
- Networking Diagram
- Oracle DDL Diagram

Press Next to move to the next screen.



☑ Use First Row as Header

Properties

Name	Type	Owner
ownedComment	String	Element
componentOfPack...	Component	Class
abstract	boolean	Class
active	boolean	Class
leaf	boolean	Class
name	String	Class
owner	Element	Class
owningPackage	Package	Class
package	Package	Class
base_Class	Class	Requirement
Text	String	Requirement
Id	String	Requirement

CSV Data

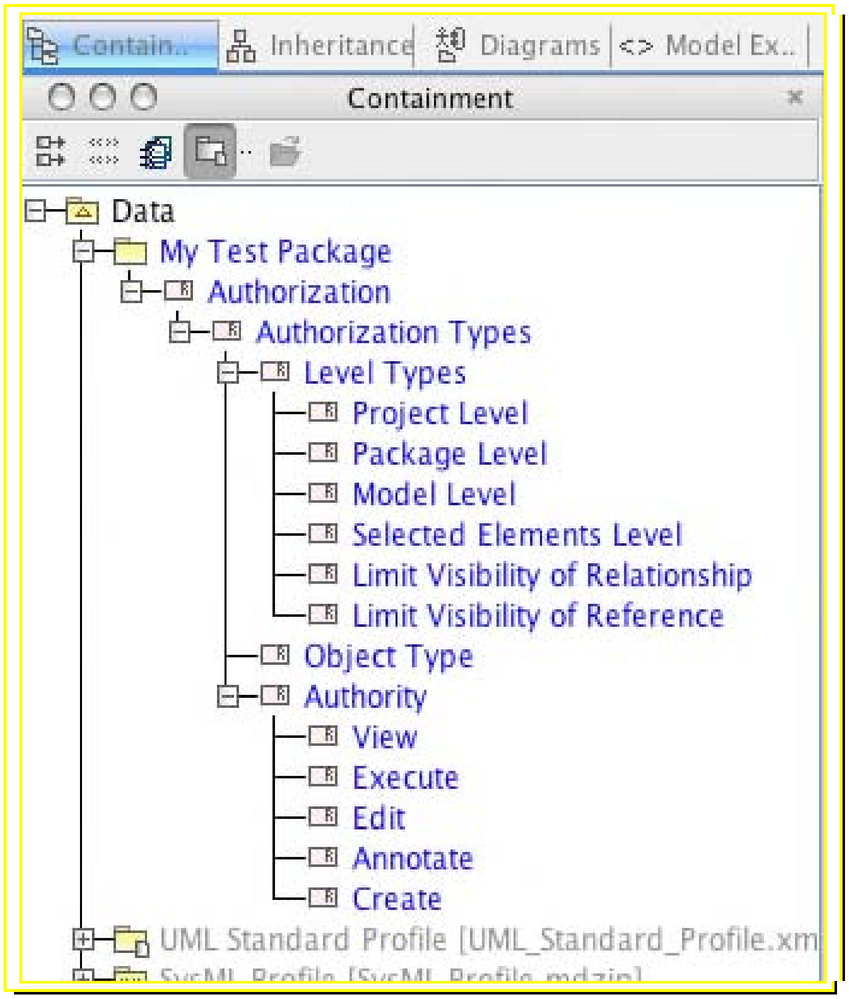
Name	Owner	DeriveFrom	Text	DiagName	De
Authorization			Authorizat...		
Authorization Types	Authorizat...		Types of ...	My Auth T...	
Level Types	Authorizat...			My Auth T...	
Object Type	Authorizat...		Type of it...	My Auth T...	
Authority	Authorizat...	Object Type	Type of A...	My Auth T...	De

Property Maps

Property Name	Owner	Column Index
name	Class	0
owner	Class	1
Text	Requirement	3

Key property: name

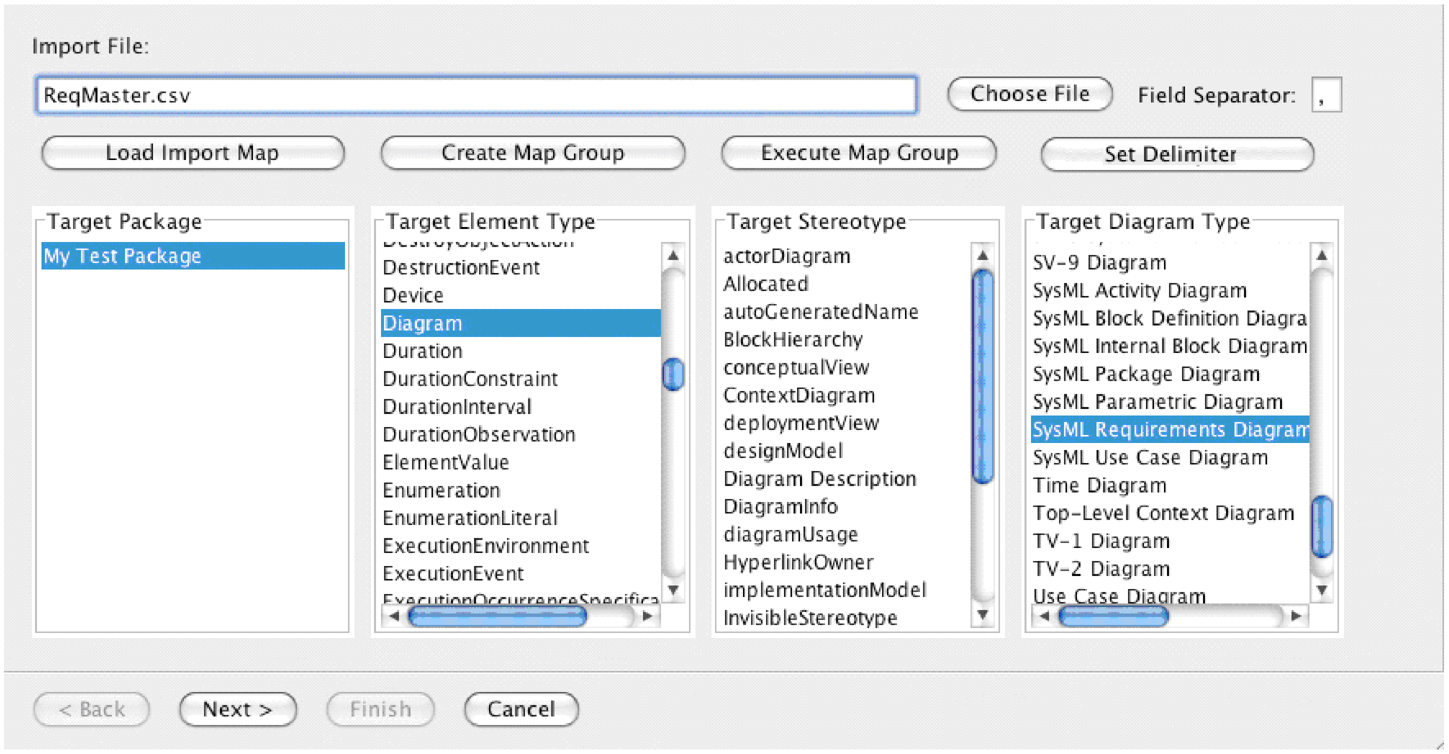
Notice that the Owner column for the Authorization element is blank. This is the only element that is owned by the package. Leaving the value blank causes the plug-in to substitute the package. Now, the containment area looks like this:



```
Data
  My Test Package
    Authorization
      Authorization Types
        Level Types
          Project Level
          Package Level
          Model Level
          Selected Elements Level
          Limit Visibility of Relationship
          Limit Visibility of Reference
        Object Type
        Authority
          View
          Execute
          Edit
          Annotate
          Create
  UML Standard Profile [UML_Standard_Profile.xm
```

With that done, we will now create the diagrams.

Start the plug-in.


Import File: ReqMaster.csv

Field Separator: ,

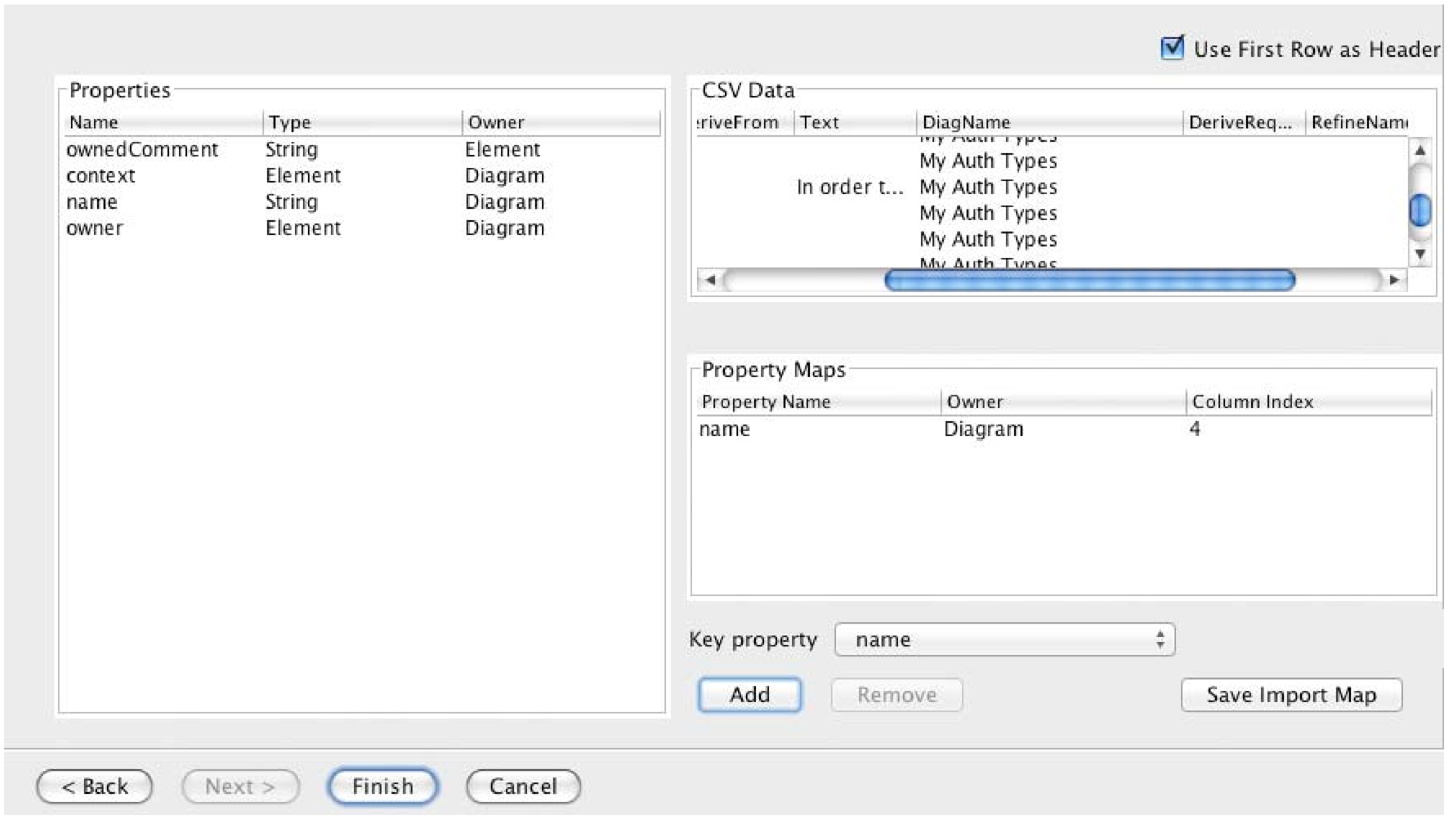
Target Package: My Test Package

Target Element Type: DestructionEvent, Device, Diagram, Duration, DurationConstraint, DurationInterval, DurationObservation, ElementValue, Enumeration, EnumerationLiteral, ExecutionEnvironment, ExecutionEvent (selected: Diagram)

Target Stereotype: actorDiagram, Allocated, autoGeneratedName, BlockHierarchy, conceptualView, ContextDiagram, deploymentView, designModel, Diagram Description, DiagramInfo, diagramUsage, HyperlinkOwner, implementationModel, InvisibleStereotype

Target Diagram Type: SV-9 Diagram, SysML Activity Diagram, SysML Block Definition Diagra, SysML Internal Block Diagram, SysML Package Diagram, SysML Parametric Diagram, SysML Requirements Diagram, SysML Use Case Diagram, Time Diagram, Top-Level Context Diagram, TV-1 Diagram, TV-2 Diagram, Use Case Diagram (selected: SysML Requirements Diagram)

Press Next to move to the next screen.


Use First Row as Header [✓]

Properties

Name	Type	Owner
ownedComment	String	Element
context	Element	Diagram
name	String	Diagram
owner	Element	Diagram

CSV Data

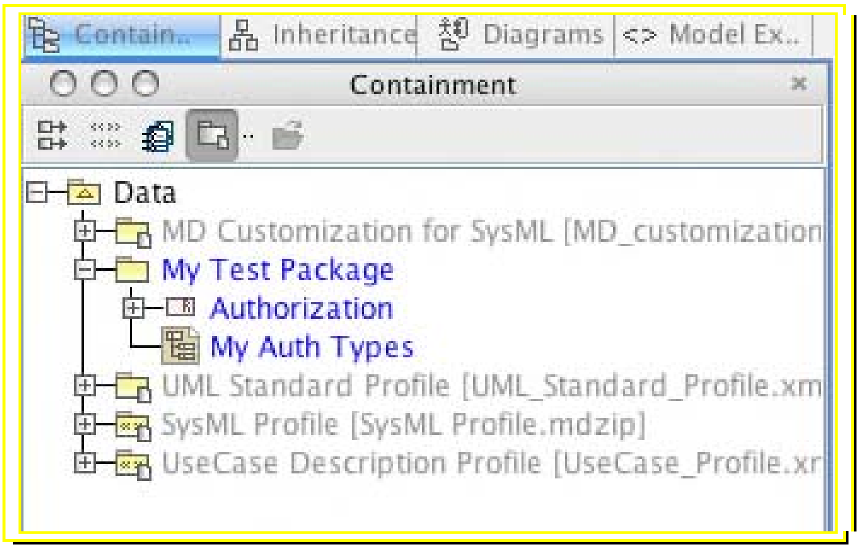
riveFrom	Text	DiagName	DeriveReq...	RefineNam
		My Auth Types		
	In order t...	My Auth Types		
		My Auth Types		
		My Auth Types		
		My Auth Types		

Property Maps

Property Name	Owner	Column Index
name	Diagram	4

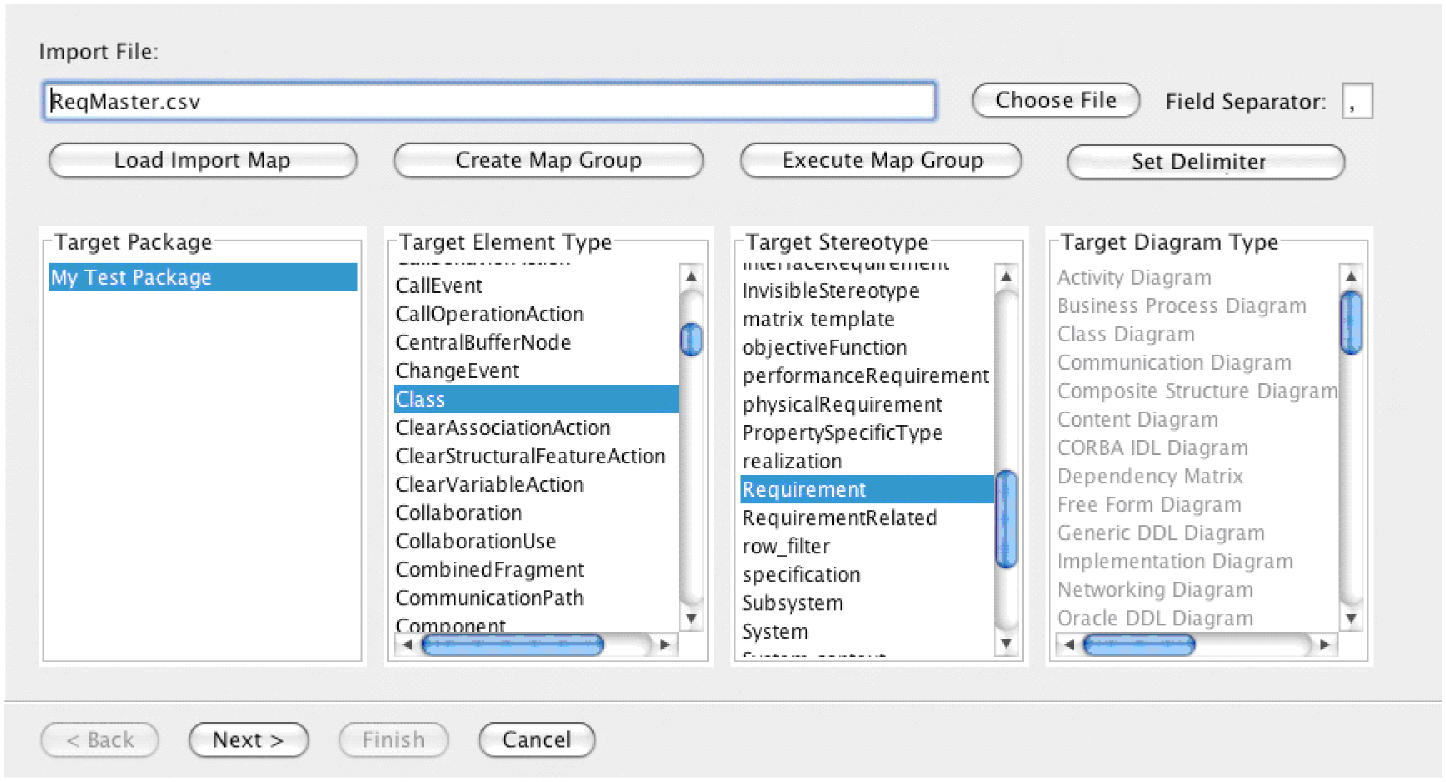
Key property: name

Press Finish and we have a new diagram in the model.



Data
- MD Customization for SysML [MD_customization
- My Test Package
  - Authorization
  - My Auth Types
- UML Standard Profile [UML_Standard_Profile.xm
- SysML Profile [SysML Profile.mdzip]
- UseCase Description Profile [UseCase_Profile.xr

Now, we'll populate the diagram with requirements. Start the plug-in.



Import File: ReqMaster.csv

Field Separator: ,

Target Package: My Test Package

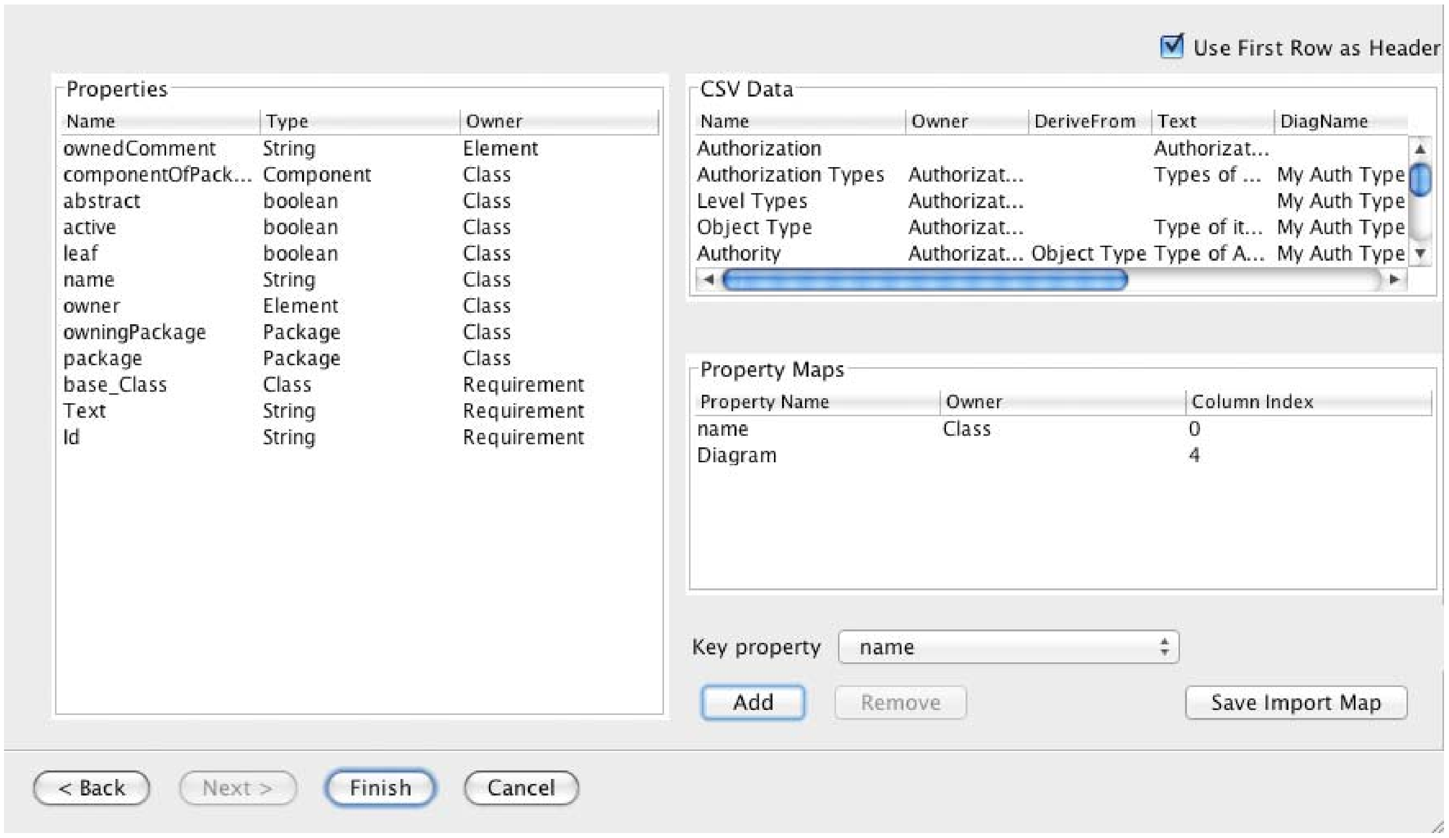
Target Element Type: CallEvent, CallOperationAction, CentralBufferNode, ChangeEvent, Class (selected), ClearAssociationAction, ClearStructuralFeatureAction, ClearVariableAction, Collaboration, CollaborationUse, CombinedFragment, CommunicationPath, Component

Target Stereotype: InvisibleStereotype, matrix template, objectiveFunction, performanceRequirement, physicalRequirement, PropertySpecificType, realization, Requirement (selected), RequirementRelated, row_filter, specification, Subsystem, System

Target Diagram Type: Activity Diagram, Business Process Diagram, Class Diagram, Communication Diagram, Composite Structure Diagram, Content Diagram, CORBA IDL Diagram, Dependency Matrix, Free Form Diagram, Generic DDL Diagram, Implementation Diagram, Networking Diagram, Oracle DDL Diagram

It may not have been necessary to reapply the Requirement stereotype, but we'll do it just in case.

Press Next to move to the next screen.



☑ Use First Row as Header

Properties

Name	Type	Owner
ownedComment	String	Element
componentOfPack...	Component	Class
abstract	boolean	Class
active	boolean	Class
leaf	boolean	Class
name	String	Class
owner	Element	Class
owningPackage	Package	Class
package	Package	Class
base_Class	Class	Requirement
Text	String	Requirement
Id	String	Requirement

CSV Data

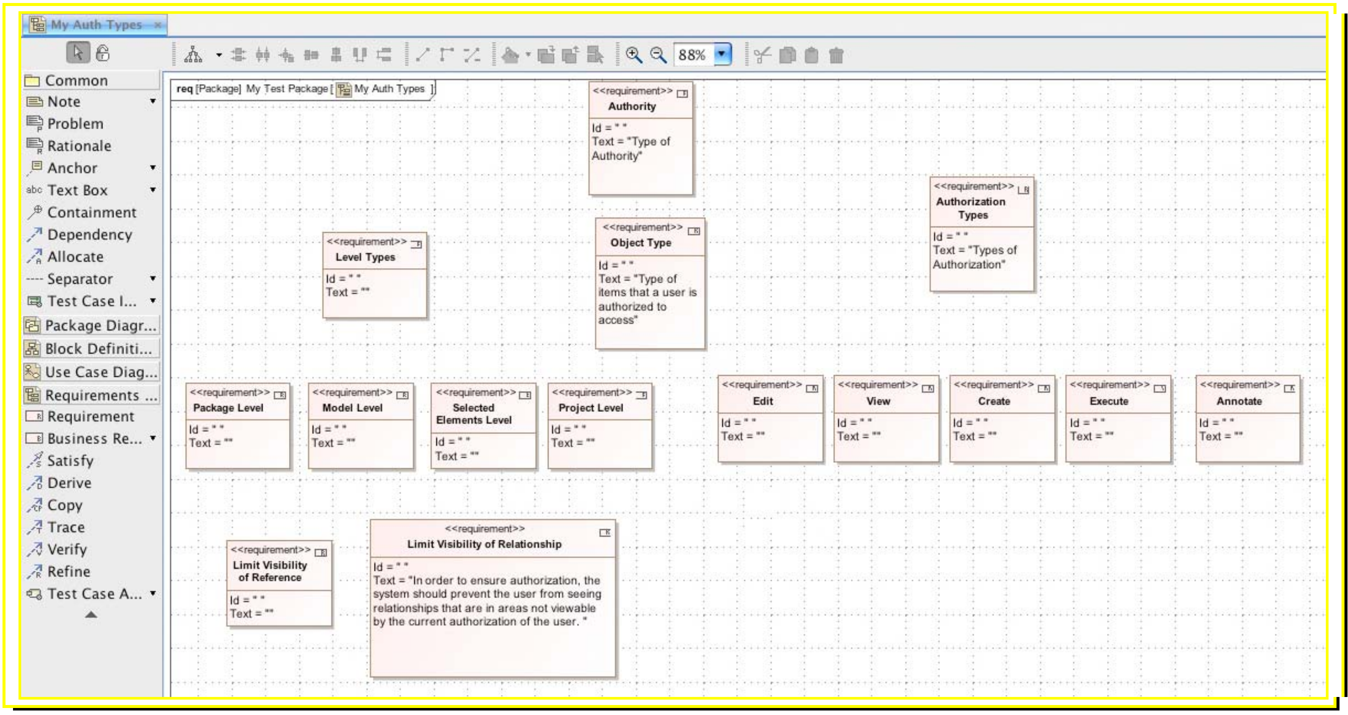
Name	Owner	DeriveFrom	Text	DiagName
Authorization			Authorizat...	
Authorization Types	Authorizat...		Types of ...	My Auth Type
Level Types	Authorizat...			My Auth Type
Object Type	Authorizat...		Type of it...	My Auth Type
Authority	Authorizat...	Object Type	Type of A...	My Auth Type

Property Maps

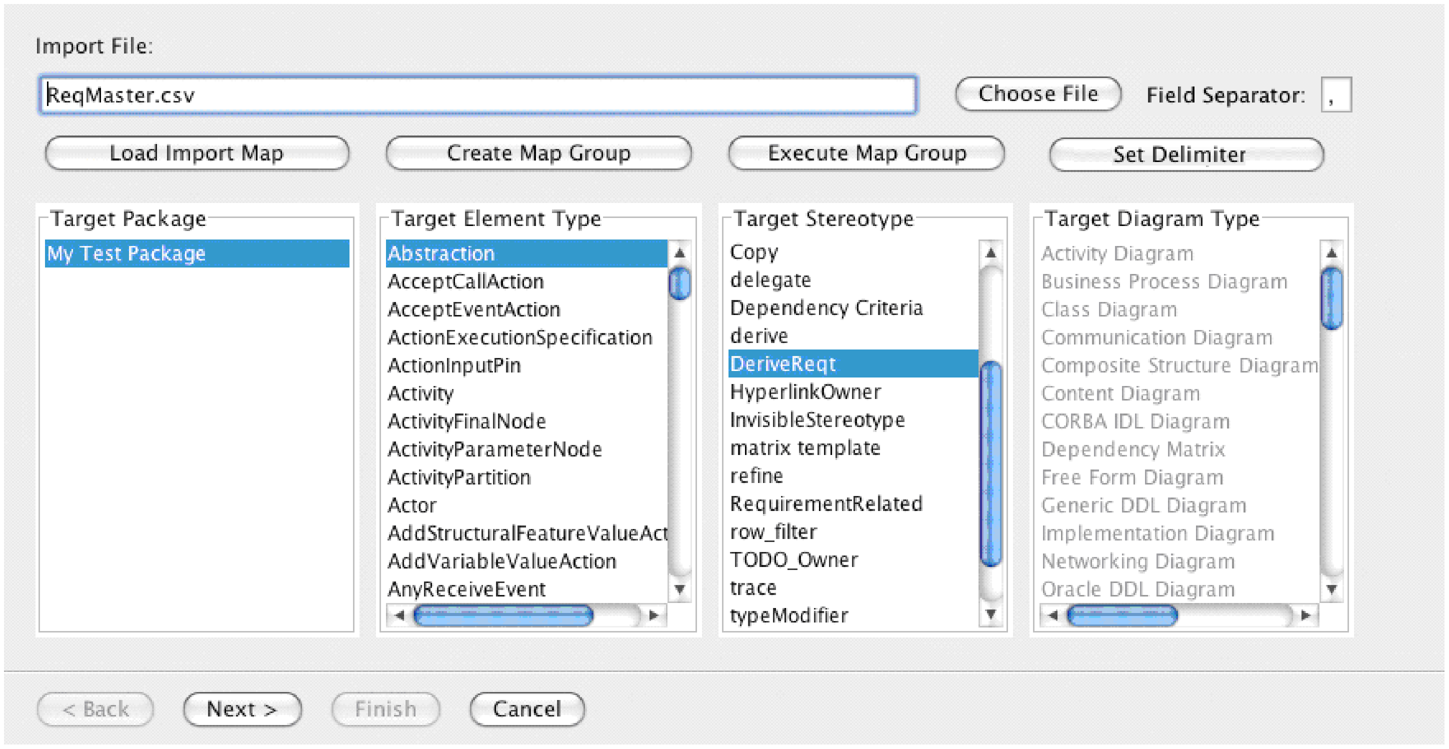
Property Name	Owner	Column Index
name	Class	0
Diagram		4

Key property: name

Press Finish and look what we have on our diagram.


My Auth Types
88%
Common
Note
Problem
Rationale
Anchor
Text Box
Containment
Dependency
Allocate
Separator
Test Case I...
Package Diagr...
Block Definiti...
Use Case Diag...
Requirements ...
Requirement
Business Re...
Satisfy
Derive
Copy
Trace
Verify
Refine
Test Case A...
req [Package] My Test Package [My Auth Types]
<<requirement>> Authority
Id = " "
Text = "Type of Authority"
<<requirement>> Authorization Types
Id = " "
Text = "Types of Authorization"
<<requirement>> Level Types
Id = " "
Text = ""
<<requirement>> Object Type
Id = " "
Text = "Type of items that a user is authorized to access"
<<requirement>> Package Level
Id = " "
Text = ""
<<requirement>> Model Level
Id = " "
Text = ""
<<requirement>> Selected Elements Level
Id = " "
Text = ""
<<requirement>> Project Level
Id = " "
Text = ""
<<requirement>> Edit
Id = " "
Text = ""
<<requirement>> View
Id = " "
Text = ""
<<requirement>> Create
Id = " "
Text = ""
<<requirement>> Execute
Id = " "
Text = ""
<<requirement>> Annotate
Id = " "
Text = ""
<<requirement>> Limit Visibility of Reference
Id = " "
Text = ""
<<requirement>> Limit Visibility of Relationship
Id = " "
Text = "In order to ensure authorization, the system should prevent the user from seeing relationships that are in areas not viewable by the current authorization of the user. "

You can, however, import other types of relationships between requirements. Bring up the plug-in once more.


Import File:

ReqMaster.csv

Field Separator: ,

Target Package
- My Test Package

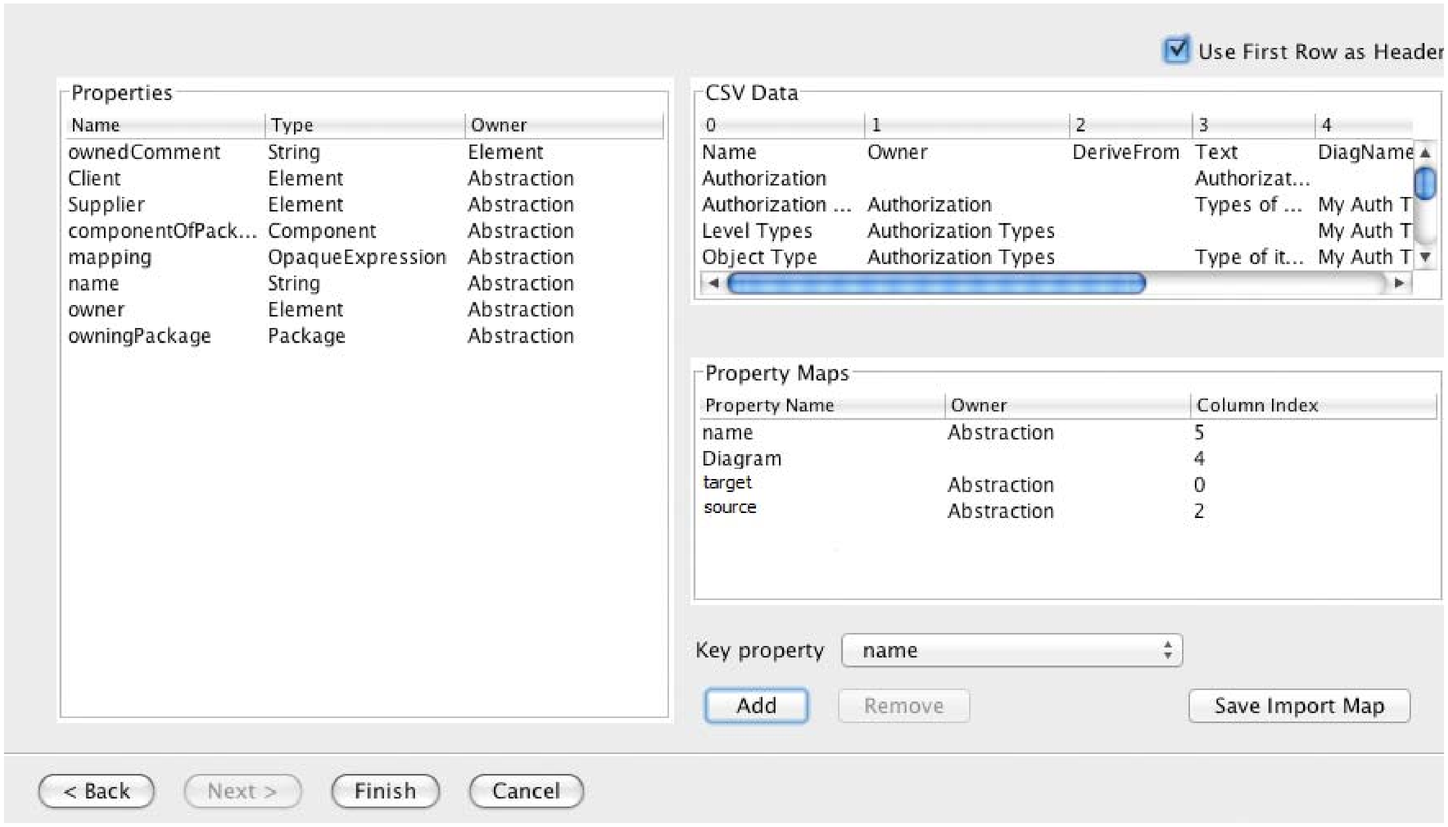
Target Element Type
- Abstraction
- AcceptCallAction
- AcceptEventAction
- ActionExecutionSpecification
- ActionInputPin
- Activity
- ActivityFinalNode
- ActivityParameterNode
- ActivityPartition
- Actor
- AddStructuralFeatureValueAct
- AddVariableValueAction
- AnyReceiveEvent

Target Stereotype
- Copy
- delegate
- Dependency Criteria
- derive
- DeriveReqt
- HyperlinkOwner
- InvisibleStereotype
- matrix template
- refine
- RequirementRelated
- row_filter
- TODO_Owner
- trace
- typeModifier

Target Diagram Type
- Activity Diagram
- Business Process Diagram
- Class Diagram
- Communication Diagram
- Composite Structure Diagram
- Content Diagram
- CORBA IDL Diagram
- Dependency Matrix
- Free Form Diagram
- Generic DDL Diagram
- Implementation Diagram
- Networking Diagram
- Oracle DDL Diagram

Notice we apply the DeriveReqt stereotype to the Abstraction element.

Press Next to move to the next screen.



☑ Use First Row as Header

Properties

Name	Type	Owner
ownedComment	String	Element
Client	Element	Abstraction
Supplier	Element	Abstraction
componentOfPack...	Component	Abstraction
mapping	OpaqueExpression	Abstraction
name	String	Abstraction
owner	Element	Abstraction
owningPackage	Package	Abstraction

CSV Data

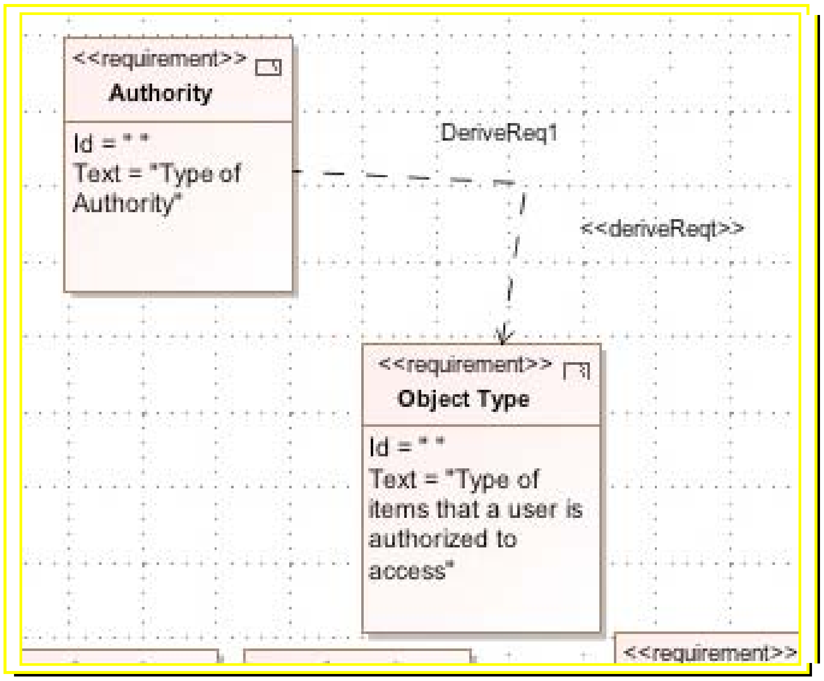
0	1	2	3	4
Name	Owner	DeriveFrom	Text	DiagName
Authorization			Authorizat...	
Authorization ...	Authorization		Types of ...	My Auth T
Level Types	Authorization Types			My Auth T
Object Type	Authorization Types		Type of it...	My Auth T

Property Maps

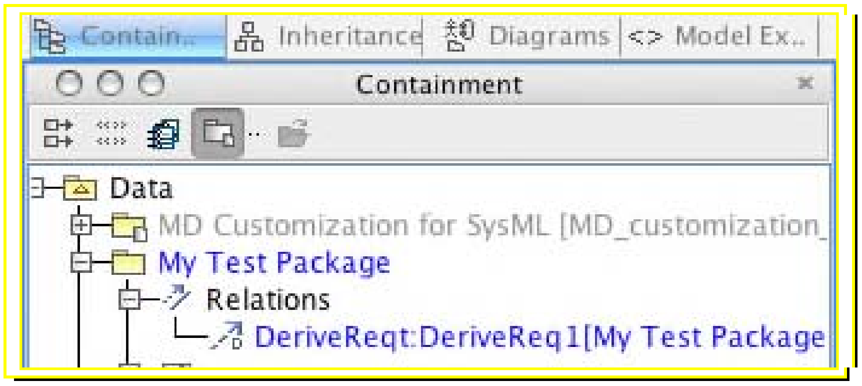
Property Name	Owner	Column Index
name	Abstraction	5
Diagram		4
target	Abstraction	0
source	Abstraction	2

Key property: name

Press Finish and we can see the effects on the diagram:



And in the containment area:



Data
MD Customization for SysML [MD_customization_
My Test Package
Relations
DeriveReqt:DeriveReq1[My Test Package