# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table of Contents</td>
<td>2</td>
</tr>
<tr>
<td>Introduction</td>
<td>4</td>
</tr>
<tr>
<td>What is a Comma Separated Values File?</td>
<td>4</td>
</tr>
<tr>
<td>How Do I Get My Data into Comma Separated Values format?</td>
<td>4</td>
</tr>
<tr>
<td>Before You Begin</td>
<td>4</td>
</tr>
<tr>
<td>Mapping Attributes</td>
<td>8</td>
</tr>
<tr>
<td>Data Conversions</td>
<td>8</td>
</tr>
<tr>
<td>The Owner Attribute</td>
<td>9</td>
</tr>
<tr>
<td>Key property</td>
<td>9</td>
</tr>
<tr>
<td>Adding Importing Elements to Diagrams</td>
<td>9</td>
</tr>
<tr>
<td>Creating Relationships</td>
<td>9</td>
</tr>
<tr>
<td>Saving Your Choices</td>
<td>9</td>
</tr>
<tr>
<td>Saving Maps</td>
<td>10</td>
</tr>
<tr>
<td>Saving Map Groups</td>
<td>11</td>
</tr>
<tr>
<td>Creating DoDAF And Other Custom Model Elements</td>
<td>12</td>
</tr>
<tr>
<td>Examples</td>
<td>13</td>
</tr>
<tr>
<td>Creating and Populating Class Diagrams</td>
<td>13</td>
</tr>
<tr>
<td>Creating Classes into Diagram</td>
<td>15</td>
</tr>
<tr>
<td>Creating Associations into Diagram</td>
<td>19</td>
</tr>
<tr>
<td>Creating and Populating State Diagrams</td>
<td>21</td>
</tr>
<tr>
<td>Creating and Populating Activity Diagrams</td>
<td>32</td>
</tr>
<tr>
<td>Creating Classes into Diagram</td>
<td>37</td>
</tr>
<tr>
<td>Creating Operations into Classes</td>
<td>40</td>
</tr>
<tr>
<td>Creating Control Flows between Operations</td>
<td>45</td>
</tr>
<tr>
<td>Creating and Populating DoDAF Diagrams</td>
<td>49</td>
</tr>
<tr>
<td>Creating Operational Nodes</td>
<td>54</td>
</tr>
<tr>
<td>Creating Needlines</td>
<td>56</td>
</tr>
<tr>
<td>Prototyping Elements in a Test Model Before you Begin</td>
<td>57</td>
</tr>
<tr>
<td>Using one CSV File to Import Multiple Types of Data</td>
<td>60</td>
</tr>
<tr>
<td>Custom Model Elements Revisited: Requirements</td>
<td>70</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>First Name</th>
<th>Last Name</th>
<th>City</th>
<th>State</th>
<th>Date Of Birth</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Wilson</td>
<td>Miami</td>
<td>FL</td>
<td>04-22-1968</td>
</tr>
<tr>
<td>Mary</td>
<td>Kelley</td>
<td>Newark</td>
<td>NJ</td>
<td>07-18-1977</td>
</tr>
</tbody>
</table>

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

![Excel spreadsheet example](image)

Export this to CSV Format.
Now, bring up the CSV Import Plug-in.

Press Next to move to the next screen.
Press Finish and look at your MagicDraw containment window.

![Containment Window]

Creating Classes into Diagram

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

![Spreadsheet Data]

Then, we run the import plug-in.
Press Next to move to the next screen.

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

Then, bring up the import plug-in

Press Next to move to the next screen.
Again, your containment area reflects the new additions to your model:
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
Bring up the plug-in.

Press Next to move to the next screen.
### Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>ownedComment</td>
<td>String</td>
<td>Element</td>
</tr>
<tr>
<td>componentOfPackage</td>
<td>Component</td>
<td>StateMachine</td>
</tr>
<tr>
<td>abstract</td>
<td>boolean</td>
<td>StateMachine</td>
</tr>
<tr>
<td>active</td>
<td>boolean</td>
<td>StateMachine</td>
</tr>
<tr>
<td>leaf</td>
<td>boolean</td>
<td>StateMachine</td>
</tr>
<tr>
<td>name</td>
<td>String</td>
<td>StateMachine</td>
</tr>
<tr>
<td>owner</td>
<td>Element</td>
<td>StateMachine</td>
</tr>
<tr>
<td>owningPackage</td>
<td>Package</td>
<td>StateMachine</td>
</tr>
<tr>
<td>package</td>
<td>Package</td>
<td>StateMachine</td>
</tr>
<tr>
<td>reentrant</td>
<td>boolean</td>
<td>StateMachine</td>
</tr>
<tr>
<td>typesForSource</td>
<td>Class</td>
<td>Customization</td>
</tr>
<tr>
<td>representationText</td>
<td>String</td>
<td>Customization</td>
</tr>
<tr>
<td>keyword</td>
<td>String</td>
<td>Customization</td>
</tr>
<tr>
<td>allowedRelations...</td>
<td>Class</td>
<td>Customization</td>
</tr>
<tr>
<td>disallowedRelation.</td>
<td>Class</td>
<td>Customization</td>
</tr>
<tr>
<td>index</td>
<td>boolean</td>
<td>Customization</td>
</tr>
<tr>
<td>superTypes</td>
<td>Element</td>
<td>Customization</td>
</tr>
<tr>
<td>baseClass</td>
<td>Class</td>
<td>Customization</td>
</tr>
<tr>
<td>standardExpertC...</td>
<td>String</td>
<td>Customization</td>
</tr>
<tr>
<td>hiddenOwnedType...</td>
<td>Class</td>
<td>Customization</td>
</tr>
<tr>
<td>suggestedOwnedType</td>
<td>String</td>
<td>Customization</td>
</tr>
</tbody>
</table>

### CSV Data

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMach1</td>
<td>Keyword1</td>
<td>State Machine 1</td>
</tr>
<tr>
<td>SMach2</td>
<td>Keyword2</td>
<td>State Machine 2</td>
</tr>
</tbody>
</table>

### Property Maps

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Owner</th>
<th>Column Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>StateMachine</td>
<td>0</td>
</tr>
<tr>
<td>keyword</td>
<td>Customization</td>
<td>1</td>
</tr>
<tr>
<td>ownedComment</td>
<td>Element</td>
<td>2</td>
</tr>
</tbody>
</table>

Key property: name
Press Finish and your containment area should look like this:

Step 2: Create State Machine Diagrams
Step 3: Bring up the plug-in

Step 4: Press Next to move to the next screen.
Step 5: Press Finish and your containment area should look like this:

![Containment Area Image]

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.
Step 8. Import them

Step 9. Press Next to move to the next screen.
Step 10. Press Finish and review your containment area.
See them populated on the State Diagrams.

Now, we will import transitions. First, let’s create the data:
Now, we'll import it. Bring up the plug-in.

Press Next to move to the next screen.
Press Finish and the containment area will look like this:
A Quick look at StateDiag1 reveals this:

Creating and populating Activity Diagrams

First, we’ll start with a clean containment area:
Now, we'll import some activities, since activity diagrams are owned by activity elements:
Bring up the plug-in:

Press Next to go the next screen.
Press Finish and your containment area should look like this:

Now, we’ll create a couple of Activity Diagrams. As always, we start with CSV data:
Now, we’ll bring up the plug-in and import the data:

Press Next to go to the next screen.
Press Finish.

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

![Containment Area](image)

**Creating Classes into Diagram**

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

![Class Table](image)
Start the plug-in

Press Next to go the next screen.
After pressing Finish, your containment area reflects the changes:

We can also see the changes reflected on the diagrams:
Creating Operations into Classes

Now, we will create some operations inside those classes.
And import them:

Press Next to go the next screen.
Press Finish and we see the changes reflected in the containment area:

Now, we’ll put these Operations on the diagram:
Start the plug-in.

Press Next to go the next screen.
After pressing Finish, the containment area gets updated...
Creating Control Flows between Operations

Now, we’ll tie these operations together with some control flows:
Start the plug-in.

Press Next to go the next screen.
After pressing Finish, the changes are reflected in the containment area:
And on the diagrams:

Creating and Populating DoDAF Diagrams

First, we'll start with a fresh containment area:
Now, we'll select to use the DoDAF module:

Press Next to go the next
screen.
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.

Press Next to go the next screen.
After pressing Finish, we see that several OV-2 diagrams have been created:

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.

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.

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:

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:

Columns A, B, and C contain data about some Class elements we want to import. Columns D through 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:
Now, we bring up the plug-in to create the diagrams:

Press Next to move to the next screen.
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:

![Containment](image)

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.

Press Next to move to the next screen.
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.

Press Next to move to the next screen.
After pressing Finish, when we review the containment area, we see that the relationships have been added:
We can also see them on diagram Diag1:

![Diagram](image_url)

**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.
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:
Press Next to move to the next screen.

Press Finish to load the module. Now, start the plug-in.
Press Next to move to the next screen.

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:

With that done, we will now create the diagrams.
Start the plug-in.

Press Next to move to the next screen.

Press Finish and we have a new diagram in the model.
Now, we'll populate the diagram with requirements. Start the plug-in.

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.
Press Finish and look what we have on our diagram.

You can, however, import other types of relationships between requirements. Bring up the plug-in once more.
Notice we apply the DeriveReqt stereotype to the Abstraction element.
Press Next to move to the next screen.

Press Finish and we can see the effects on the diagram:
And in the containment area: