# Creating the UML Model for caCORE SDK

The SDK Code Generator is based upon a Model-Driven Architecture[[1]](#footnote-2) (MDA) which supports the implementation of the following scenarios specified via a UML model:

* Modeling of class attributes including :
  + A simple (primitive) attribute, such as an *integer* or *string*;
  + A collection of simple (primitive) attributes; and,
  + An identifier attribute that is named something other than the default (*ID)* in the Logical (Object) Model.
* Modeling of class associations, including:
  + Uni- and bi-directional associations;
  + Many-to-Many, Many-to-One, One-to-Many, and One-to-One associations;
  + Associations that use a Join Table;
  + Associations that *don’t* use a Join Table;
* Modeling of inheritance that is implemented using:
  + One table per class in inheritance hierarchy
  + One table per inheritance hierarchy
  + One table per inheritance hierarchy, with a separate table for leaf-level child class(es)

The caCORE SDK distribution provides a sample model that demonstrates how these scenarios, and many others, can be modeled in a manner that is understood by the SDK Code Generator. The sample model is intended to be used as a reference when creating your own model. The sample model is located within the *\models* directory of the SDK distribution, and has been implemented in both Enterprise Architect and ArgoUML. The name of the sample model project file is, respectively:

* Enterprise Architect: *SDKTestModel.EAP*
* ArgoUML: *sdk.uml*

The following sections describe how to perform various modeling activities using both the Enterprise Architect (EA) and ArgoUML modeling tools.

## Creating a New Project

To create a new object model project file, follow these steps:

1. In EA, open the *SDKEATemplate.EAP* baseline file provided in the \models directory of the SDK distribution. This file already contains the base *Logical View*, *Data Model*, and *Logical (Object) Model* packages, as well as classes representing the wrapper Java primitive type classes, as shown below:

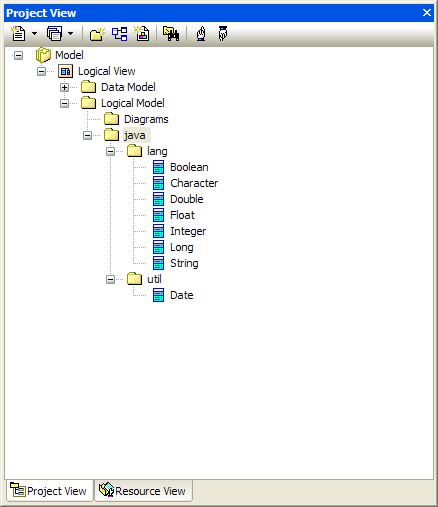


Figure EA Project View Browser

In ArgoUML, open the *SDKArgoTemplate.EAP* baseline file provided in the \models directory of the SDK distribution. This file already contains the base *Logical View*, *Data Model*, and *Logical (Object) Model* packages, classes representing the wrapper Java primitive type classes, *Tag Definitions (TD)* for all the possible Tag Value types, and *DataTypes* as shown below:

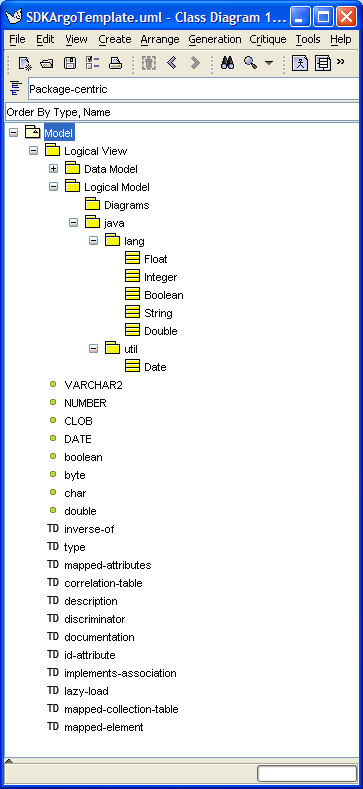
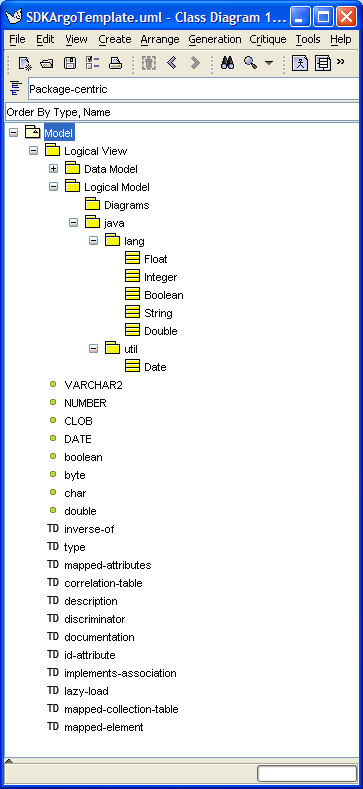
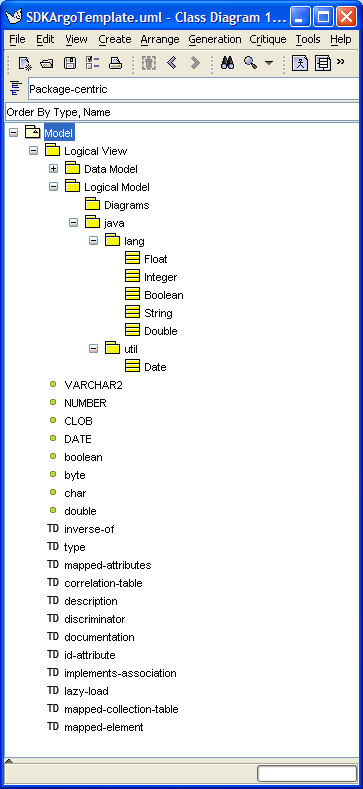
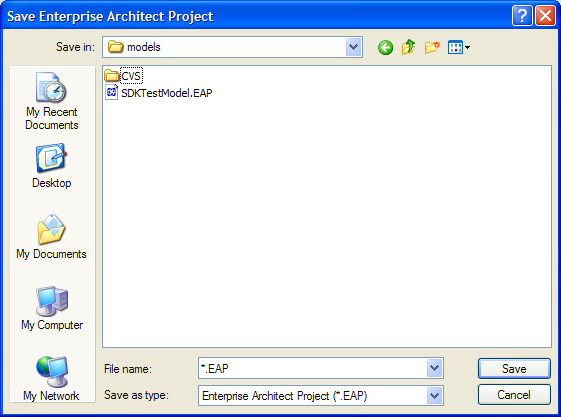
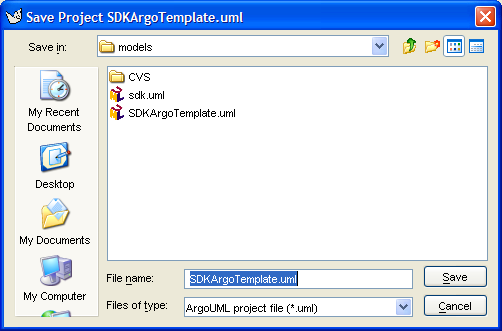
  

Figure ArgoUML Explorer Pane Showing Packages/Classes, Tag Definitions, Data Types

1. Select the *File | Save Project As* option.
2. In EA, the Save Enterprise Architect Project dialog will appear, as shown below:



In ArgoUML, the Save Project dialog will appear, as shown below:



1. Enter a new project name in the *File name* field.
2. Click the *Save* button.

Alternatively, either of the baseline template files, *SDKEATemplate.EAP* or *SDKArgoTemplate.uml*, can be copied and renamed. The new project file is now ready for use in creating the object and data model.

## Creating Classes/Tables

UML *Class* elements are used to represent both *Logical (object) Model* classes and *Data Model* classes (tables). Object classes are typically created using a package hierarchy within the Logical Model package, while Data Model classes (tables) are created directly within the Data Model package without the use of a package hierarchy.

### Creating a Logical Model Package Structure

To add a package structure to the *Logical Model*, follow these steps:

1. In EA, select the *Logical Model* package, as shown below:

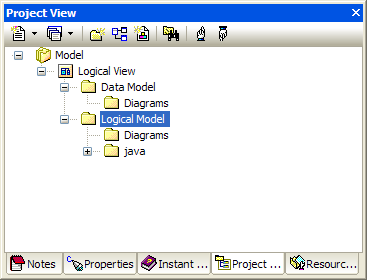


Figure EA Project Browser

In ArgoUML, select the *Logical Model* package, as shown below:

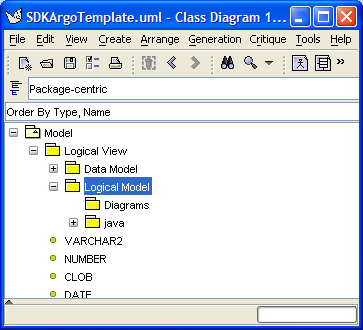


Figure ArgoUML Explorer[[2]](#footnote-3) Pane

1. In EA, right click and select the *Add | Add Package* menu OR go to the *Project | Add Package* submenu. In ArgoUML, right click and select the *Add Package* menu option.
2. In EA, The following dialog will appear, as shown below:

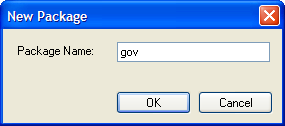


Figure EA New Package Dialog

In ArgoUML, the *Properties* tab in the *Detail* pane will become active for the new package, as shown below:

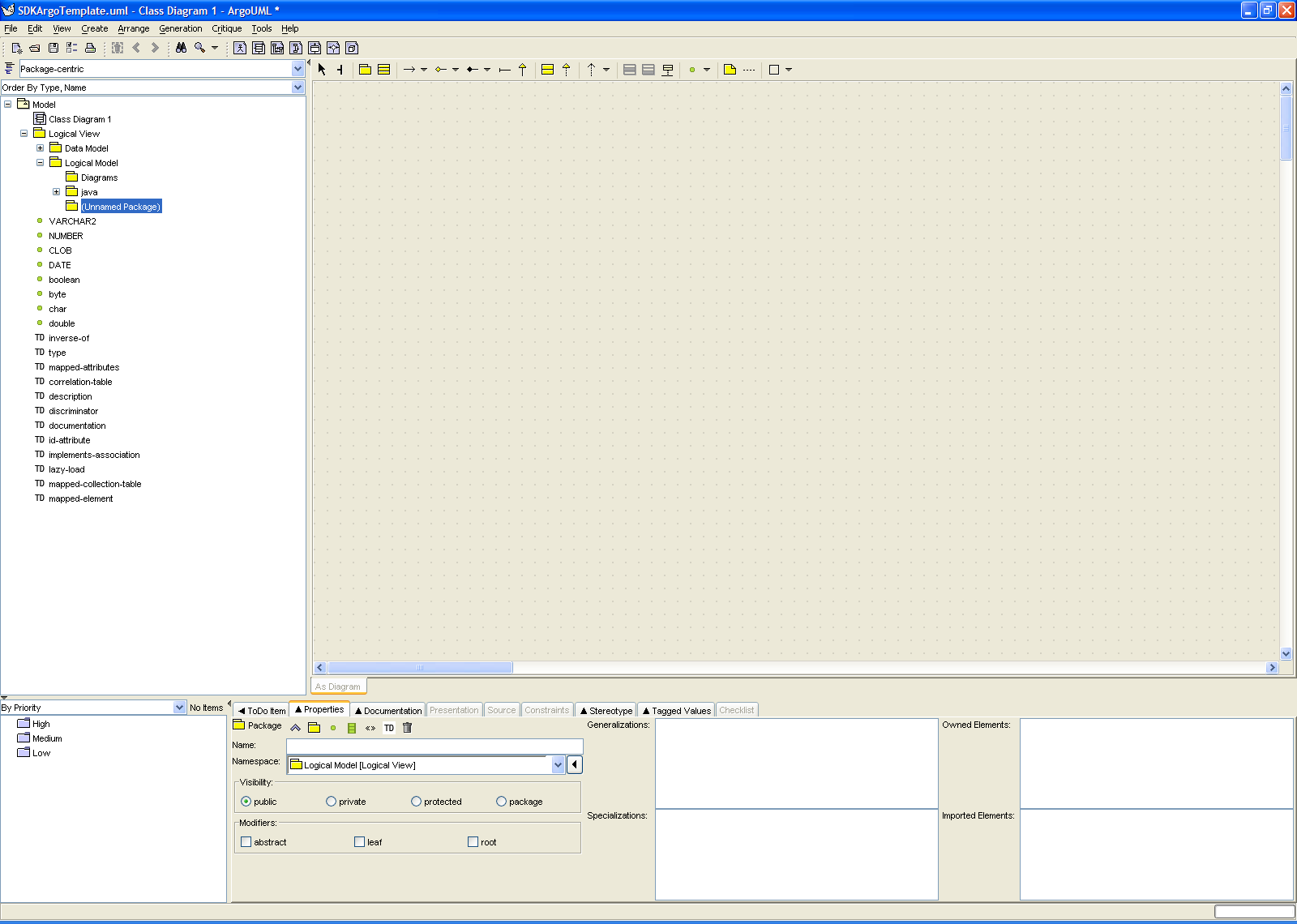


Figure ArgoUML Package Detail Pane, Properties[[3]](#footnote-4) Tab

1. In EA, enter a package (folder) name, and then click *OK.* In ArgoUML, click the save_project *Save Project* icon on the menu bar, or, alternatively, the C*trl-S* keys.

|  |  |
| --- | --- |
| NOTE: | Package names should follow Java package naming conventions; i.e., Java packages are defined using a hierarchical, *lowercase*, naming pattern, with levels in the hierarchy separated by periods (.) . Furthermore, package names are typically the organization’s domain name backwards. An example, taken from the SDK sample model, is *gov.nih.nci.cacoresdk.domain.*  When implemented within EA, each period designates the end of one package level, and the start of a new package level (termed a *subpackage*). Each package/subpackage needs to be created individually. I.e., no period(s) should be used when specifying a package name in the *New Package* dialog. Thus the fully qualified package *gov.nih.nci.cacoresdk.domain* requires a total of five (5) packages to be created within the model, one for each of the package levels. Each package is nested within the higher level one. |

1. Repeat steps 2-4 until the fully qualified package hierarchy has been created. To create a package within another package (as a sub-package/folder), select the existing package first, and then follow steps 2-4 above. The following diagram *(EA only)* shows most of the package hierarchy created in this manner for the SDK sample model:

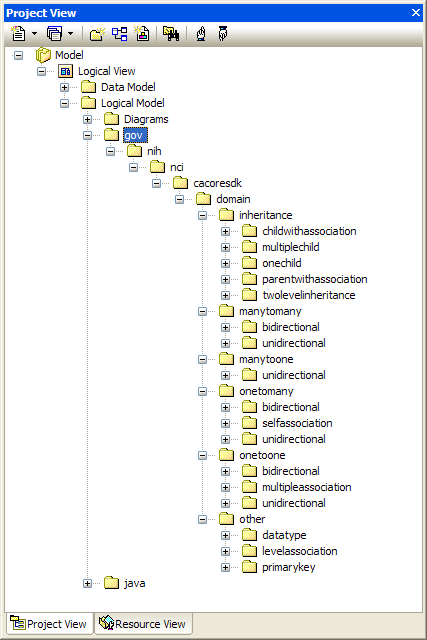


Figure EA SDK Sample Model Packages

### Creating a Logical (Object) Model Class

To add a Logical Model class to a package, follow these steps:

1. In the EA *Project Browser*, or the ArgoUML *Explorer* pane, select the desired *Logical Model* *package* to which the class should be added.
2. In EA, right click and select the *Add | Add Element* menu OR go to the *Project | Add Element* submenu.

In ArgoUML, new classes are added in the context of a class diagram within the selected package. If need be, click on a class diagram within the package to open/activate it, or create a new class diagram within the package if none exists. Select the *New Class* icon found at the top of the diagram *Editing[[4]](#footnote-5)* pane then draw a new class within the diagram. This will effectively create a new class within the selected package. Alternatively, click on an existing class within the selected package. The *Property* tab of the *Detail* pane will display the properties for the selected class. It will also display the *New Class* icon. Click on this icon.

1. In EA, the Insert New Element dialog will appear, as shown below:

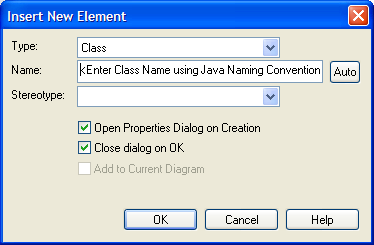


Figure EA Insert New Element Dialog

In ArgoUML, the *Properties* tab in the *Detail* pane will become active for the new class, as shown below:

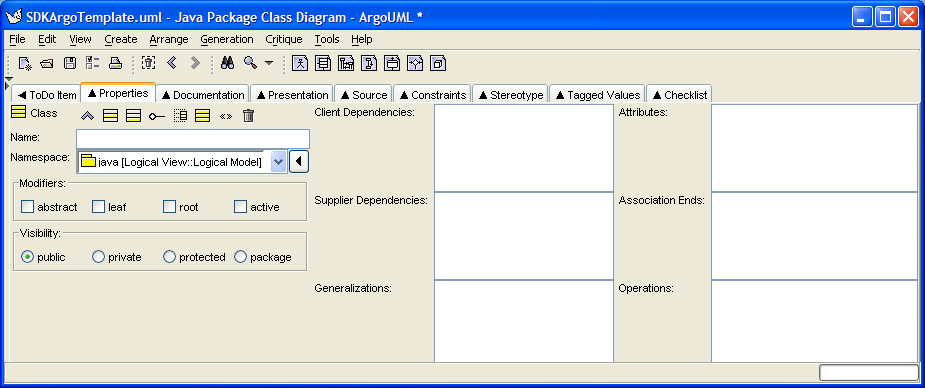


Figure ArgoUML Class Detail Pane, Properties Tab

1. In EA, set the *Insert New Element* options as follows:

| **Insert New Element Option** | **Description** |
| --- | --- |
| Type | Select *Class* as the element type from the drop down list. |
| Name | Enter a class name according to the Java class naming conventions; i.e., class names should start with a capital letter, with embedded words capitalized. |
| Stereotype | Leave blank for Logical (object) Model classes. |
| Open Property Dialog | Check the *Open Property Dialog* option if you want the *Property* dialog to open immediately after the class is created. |
| Close dialog on OK | Uncheck the *Close dialog on OK* option if you want to add multiple classes in one session. |

|  |  |
| --- | --- |
| NOTE: | Logical (Object) Model class names should follow Java class naming conventions; i.e., class names should start with a capital letter, with embedded words capitalized. Here’s an example from the SDK sample model: *GraduateStudent*. |

In ArgoUML, enter a class name in the *Name* field according to the Java class naming conventions; i.e., class names should start with a capital letter, with embedded words capitalized. Next, select the desired package where the class should be created from the *Namespace* drop down.

1. In EA, click *OK*. If the *Open Property Dialog* was checked, the *Property* dialog will open up immediately after the class is created. The Property dialog for the SDK sample *Credit* class is shown below *(EA)*. Note that the *Stereotype* field is blank since this class represents a domain object, and not a data table:

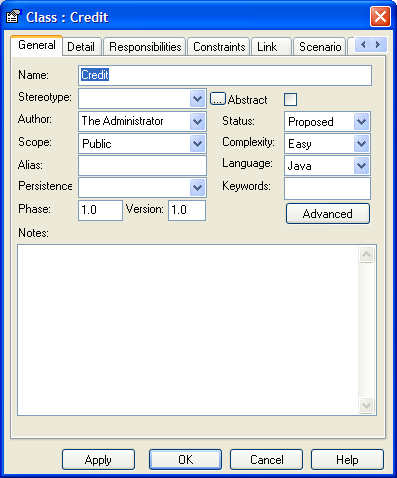


Figure EA Class Property Dialog

In ArgoUML, click the http://argouml-stats.tigris.org/documentation/manual-0.24/images/icons/save_project.gif *Save Project* icon on the menu bar, or, alternatively, the C*trl-S* keys.

See section for instructions on how to add attributes to classes.

1. Repeat steps 1-5 as needed to add other classes. In EA, if the *Close dialog on OK* option was unchecked in the *Insert New Element* dialog, additional classes can be created in the selected package by repeating steps 4-5. The following diagram shows a series of classes that have been created in the many-to-many *bidirectional* and *unidirectional* packages of the SDK Sample model:

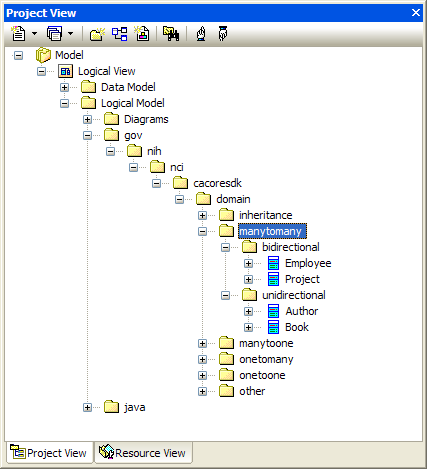


Figure EA Project View Browser Showing SDK Sample Classes

### Creating a Data Model Table

To add a Data Model (Table) class, follow these steps:

1. In the EA *Project Browser*, select the *Data Model* package, as shown below:

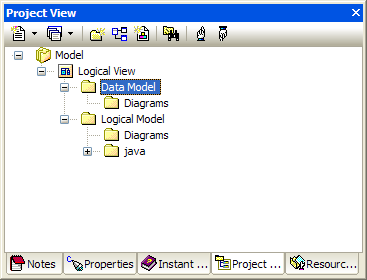


Figure EA Data Model Package

In ArgoUML *Explorer* pane*,* select the *Data Model* packageas shown below:

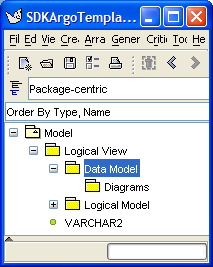


Figure ArgoUML Data Model Package

|  |  |
| --- | --- |
| NOTE: | Unlike object model classes that are created using a package hierarchy, all table classes should be created within the *Data Model* package. |

1. In EA, right click and select the *Add | Add Element* menu OR go to the *Project | Add Element* submenu.

In ArgoUML, new classes (tables) are added in the context of a class diagram within the selected package. If need be, click on a class diagram within the Data Model package to open/activate it, or create a new class diagram if none exists. Select the *New Class* icon found at the top of the diagram *Editing[[5]](#footnote-6)* pane then draw a new class within the diagram. This will effectively create a new class within the selected package. Alternatively, click on an existing class (table) within the selected package. The *Property* tab of the *Detail* pane will display the properties for the selected class. It will also display the *New Class* icon. Click on this icon.

1. In EA, the Insert New Element dialog will appear, as shown below:

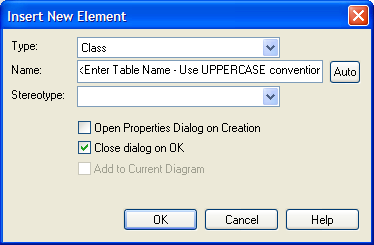


Figure EA Insert New Element (Table) Dialog

In ArgoUML, the *Properties* tab in the *Detail* pane will become active for the new table (class).

1. In EA, set the *Insert New Element* options as follows:

| **Insert New Element Option** | **Description** |
| --- | --- |
| Type | Select *Class* as the element type from the drop down list. |
| Name | Enter a table name according to the Table naming conventions; i.e., table names should be all uppercase, with embedded words separated by an underscore (\_). |
| Stereotype | Select *table* from the drop down list. |
| Open Property Dialog | Check the *Open Property Dialog* option if you want the *Property* dialog to open immediately after the table is created. |
| Close dialog on OK | Uncheck the *Close dialog on OK* option if you want to add multiple tables in one session. |

|  |  |
| --- | --- |
| NOTE: | Table names should follow Table naming conventions; i.e., table names should be all uppercase, with embedded words separated by an underscore (\_). An example from the SDK sample model: *UNDERGRADUATE\_STUDENT*. |

In ArgoUML, enter a class name in the *Name* field of the *Properties* tab according to the Java class naming conventions; i.e., class names should start with a capital letter, with embedded words capitalized. Next, select the *Data Model* package from the *Namespace* drop down, as shown below:

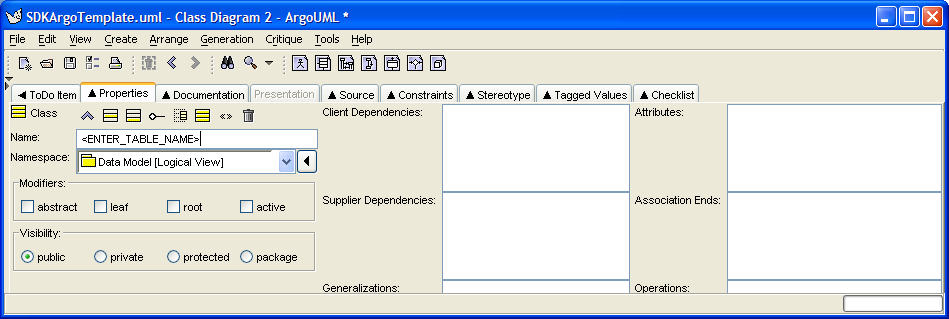


Figure ArgoUML Table (Class) Properties Tab

Finally, click on the *Stereotype* tab, select the *table* Stereotype, and apply it to the new class by clicking the **>>** icon, as shown below:

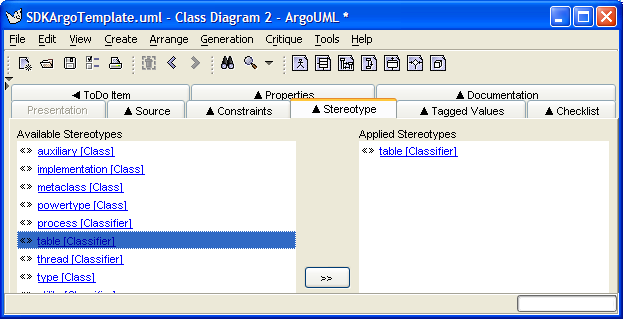
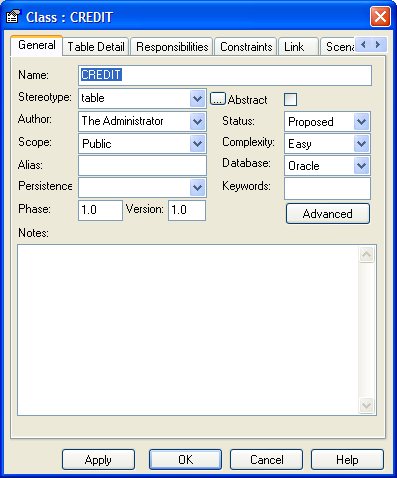


Figure ArgoUML Applying a Stereotype to a Table Class

Alternatively, select the class within a diagram, right-mouse click to bring up the context menu, and then select the *table* Stereotype from the *Apply Stereotypes* sub-menu.

1. In EA, click *OK*. If the *Open Property Dialog* was checked, the *Property* dialog will open up immediately after the class is created. The Property dialog for the SDK sample *Credit* table is shown below. Note that the *Stereotype* field is set to *table*:



In ArgoUML, click the http://argouml-stats.tigris.org/documentation/manual-0.24/images/icons/save_project.gif *Save Project* icon on the menu bar, or, alternatively, the C*trl-S* keys.

See section for instructions on how to add attributes (columns) to tables.

1. Repeat steps 1-5 as needed to add other tables. If the *Close dialog on OK* option was unchecked in the *Insert New Element* dialog, additional tables can be created in the *Data Model* package by repeating steps 4-5. The following diagram shows various tables that have been created in the *Data Model* package of the SDK sample model:

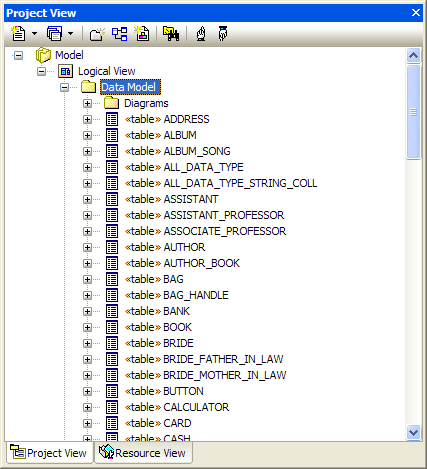


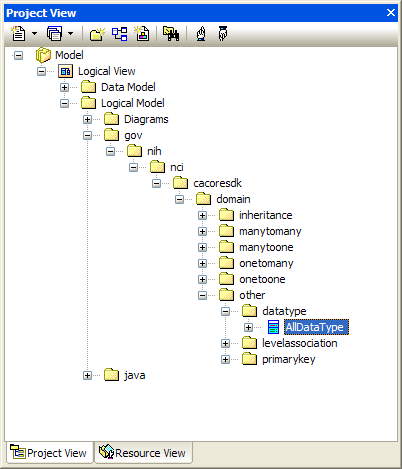
Figure EA Various Tables from the SDK Sample Model

## Creating Attributes and Data Types

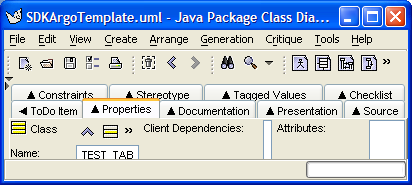
UML *Attribute* elements are used to represent both Logical (Object) Model class attributes and Data Model table columns (class attributes). Both Logical Model and Data Model class attributes can be added/modified using the same process outlined below.

To add/modify a class or table attribute, follow these steps:

1. Select the desired *Logical Model* class or *Data Model* table (class) element. The diagram below shows the Logical Model *AllDataType* class from the SDK sample model selected:



1. In EA, right click and select the *Attributes* option from the menu OR go to the *Element / Attributes* menu option.

In ArgoUML, the *Properties* tab in the *Detail* pane will become active for the selected class. Click on the  New Attribute icon.

Alternatively, select the class within a diagram, right-mouse click to bring up the context menu, and then select the *New Attribute* option from the *Add* sub-menu.

1. In EA, the Attributes dialog will appear. Shown below is the *Attributes* dialog for the Logical Model *AllDataType* class from the SDK sample model. This class illustrates all of the available primitive data types (including primitive collections) that can be assigned to a class attribute:

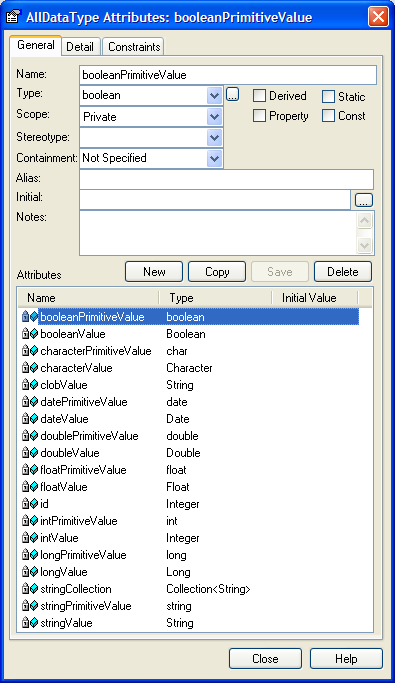


Figure Sample AllDataType Class Showing Available Attribute Primitive Data Types

In ArgoUML, the *Attribute* properties tab will become active in the *Detail* pane, as shown below:

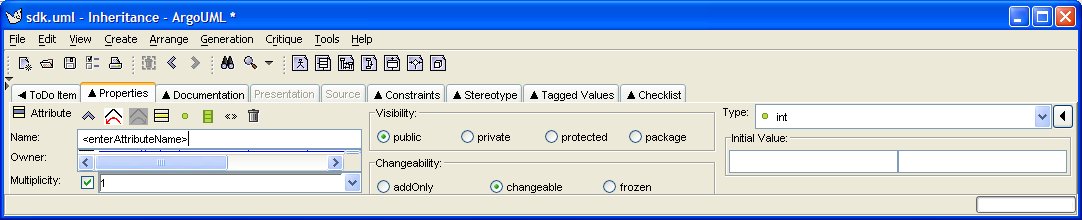


Figure ArgoUML Attribute Properties Tab

1. In EA, to *add* an attribute, click *New*, type an attribute name in the *Name* field, select a type from the *Type* drop down, and then click *Save*.

|  |  |
| --- | --- |
| NOTE: | The SDK Code Generator is only concerned with the *Name* and *Type* fields. All other fields on the EA *General* tab of the *Attributes* dialog can be ignored (left default).  Also, the SDK Code generator understands both primitive wrapper class types (e.g. Boolean) and primitive types (e.g., boolean). If a particular data type is not shown in the drop down, it can be entered (typed) into the *Type* field.  For a list of the primitive attribute types understood by the SDK Code Generator, reference the *AllDataType* class in the SDK sample model (also shown in the diagram above). Note that primitive collection types (e.g., the *stringCollection* attribute of type *Collection<String>*) are also understood as an attribute type. |

In ArgoUML, enter an attribute name in the *Name* field, and then select a type from the *Type* drop down. Click the http://argouml-stats.tigris.org/documentation/manual-0.24/images/icons/save_project.gif *Save Project* icon on the menu bar, or, alternatively, the C*trl-S* keys to save the changes.

1. To *modify* an attribute in EA, select it in the *Attributes* dialog, change the value of the *Name* and/or *Type* field, and then click *Save*.

To *modify* an attribute in ArgoUML, select the attribute in the *Explorer* pane by expanding the class to show its attributes, or click on the attribute name within the class in the *Editing* pane, if working with a diagram. The *Attribute* properties tab will become active for the attribute. Change the value of the *Name* and/or *Type* field. Click the http://argouml-stats.tigris.org/documentation/manual-0.24/images/icons/save_project.gif *Save Project* icon on the menu bar, or, alternatively, the C*trl-S* keys to save the changes.

## Performing Object Relational Mapping

The SDK Code Generator relies on information contained within custom Tag Values to generate particular system artifacts whenever the information needed cannot be derived from the UML model elements (*Class*, *Attributes*, and *Associations*) directly. Tag Values, for instance, are used to hold class/attribute documentation (comments and/or descriptions) while generating Java Docs for the object model. More importantly, however, Tag Values are used extensively when generating Hibernate Object Relational Mapping *(.hbm.xml)* files. Basically, it can be said that custom Tag Values are at the *heart* of the Logical (Object) Model-to-Data (Table) Model mapping process.

The SDK distribution provides a sample model (located within the *\models* directory) that demonstrates how various scenarios can be modeled through the use of custom Tag Values. Also, a reference table describing each of the various custom Tag Values and their usage is provided in section *.*

For those who may find that working directly with the Tag Values may be too cumbersome or error prone, please reference the **caAdapter**[[6]](#footnote-7) tool, which, among other capabilities, provides the ability to map object models to data models via a Graphical User Interface (GUI).

### Adding/Modifying Tag Values

In EA, Tag Values attached to a particular UML element (such as a *Class*, *Attribute*, or *Association*) can be added/modified via the *Tag Value* browser, which is accessible by sequentially clicking and holding down the *Ctrl-Shift-6* keys.

Given the following diagram from the SDK sample model which illustrates an association between the *Employee* and *Project* Logical Model classes:



Figure Employee-Project Asssociation Diagram

A sample of the Tag Value browser for the *Association* (line) between both classes is shown below:

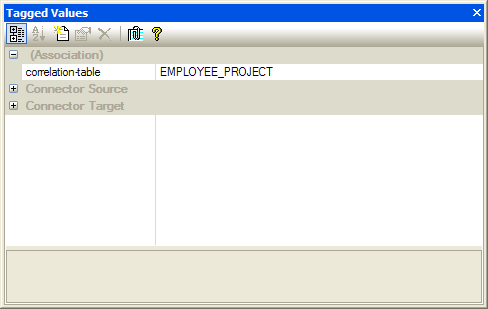


Figure EA Tag Values Browser

Once the *Tagged Values* browser is open, selecting a particular UML element (such as a *Class*, *Attribute*, or *Association*) will cause the browser to display the corresponding Tag Values attached to the selected element.

In ArgoUML, Tag Values attached to a particular UML element (such as a *Class*, *Attribute*, or *Association*) can be added/modified by first selecting the element. This will cause the *Detail* pane to become active for the selected element. Next, click on the *Tagged Values* tab to activate it.

A sample of the Tagged Values tab for the Association between the Sample SDK Employee and Project Logical Model classes is shown below:

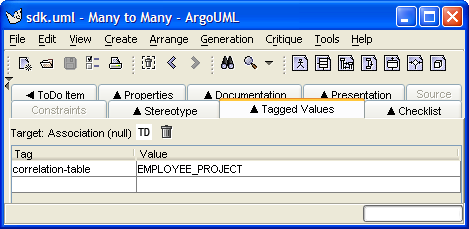


Figure ArgoUML Detail Pane, Tagged Values Tab

### SDK Custom Tag Value Descriptions

The following table lists the various Tag Values understood by the SDK Code Generator, and also describes when and where to use them:

| **Tag Value** | **Description** |
| --- | --- |
| correlation-table | A Tag Value added to an *Association* element (line) drawn between two Logical Model classes within the same diagram. The value specifies the correlation (join) table name.  Given the following *Many-to-Many* relationship diagram:    A couple of corresponding examples from the SDK sample model are provided in the table below:   |  |  |  | | --- | --- | --- | | Logical Model Class (Source) | Logical Model Class (Target) | Tag Value (correlation-table)  *Note: should be added to the Association (line) element* | | Employee | Project | EMPLOYEE\_PROJECT | | Book | Author | AUTHOR\_BOOK | |
| description | An optional Tag Value added to a *Class* or *Attribute* element to store documentation/comments for the element. The value describes the element, and is used when creating Java Docs for generated domain objects.  *Note: The description Tag Value is only used if the documentation tag value for the element is empty or does not exist.* |
| discriminator | A Tag Value added to a Data Model class *Attribute* element. The value of this tag represents the Logical Model class name that acts as the *discriminator* in situations when the parent and sub-class are persisted within the same database table. The value of the tag, if present, is placed within the *discriminator[[7]](#footnote-8)* element of the generated Hibernate mapping file.  A couple of examples from the SDK sample model is provided in the table below:   |  |  |  | | --- | --- | --- | | Data Model Class (Table) | Data Model Attribute (Column) | Tag Value (discriminator) | | SHOES | DISCRIMINATOR | gov.nih.nci.cacoresdk.domain.inheritance.childwithassociation.sametable.Shoes | | GOVERNMENT | DEMOCRATIC\_ DISCRIMINATOR | gov.nih.nci.cacoresdk.domain.inheritance.twolevelinheritance.sametable.DemocraticGovt | |
| documentation | An optional Tag Value added to a UML element to store documentation/ comments for the element. The value will be used when creating Java Docs for the generated domain object.  See also the *documentation* Tag Value. |
| id-attribute | A Tag Value added to a Logical Model class *Attribute*. The presence of the Tag Value indicates that the attribute is the class *identifier* attribute. This Tag Value is required when the identifier attribute is named something other than the default name, *id*. The value should specify the fully-qualified name of the Logical Model class that contains the attribute.  An example from the SDK sample model is provided in the table below:   |  |  |  | | --- | --- | --- | | Logical Model Class Name | Logical Model Attribute Name | Tag Value (id-attribute) | | NoIdKey | myKey | gov.nih.nci.cacoresdk.domain.other.primarykey.NoIdKey | |
| implements-association | A Tag Value added to a Data Model class *Attribute* (column). The value specifies the associated Logical Model class attribute that implements the association. The value must be specified using the following pattern: *<fully-qualified logical model class name>.< attribute name>*.  A couple of examples from the SDK sample model are provided in the table below:   |  |  |  | | --- | --- | --- | | Data Model Class (Table) | Data Model Attribute (Column) | Tag Value (implements-association) | | CARD | SUIT\_ID | gov.nih.nci.cacoresdk.domain.other.levelassociation.Card.suit | | ASSISTANT | PROFESSOR\_ID | gov.nih.nci.cacoresdk.domain.inheritance.parentwithassociation.Assistant.professor | |
| inverse-of | A Tag Value added to a Data Model class *Attribute* (column). Used to identify the inverse attribute (column) of a bi-directional association. The value specifies the corresponding inverse Logical Model class attribute, and must have the same value as the *implements-association* Tag Value of the bi-directional association. The value must be specified using the following pattern: *<fully-qualified logical model class name>.< attribute name>.*  Given the following Logical Model class diagram from the sample SDK model:    And the corresponding Data Model class diagram:    An example *inverse-of* Tag Value would be:   |  |  |  | | --- | --- | --- | | Data Model Class (Table) | Data Model Attribute (column) | Tag Value (inverse-of) | | ALBUM\_ SONG | SONG\_ID | gov.nih.nci.cacoresdk.domain.manytoone.unidirectional.withjoin.Song.album |   This indicates that the SONG\_ID attribute (column) is the inverse side of the Song/Album bi-directional association implemented by *album* attribute of the *Song* class.   |  |  | | --- | --- | | NOTE: | When adding an *inverse-of* value to a Data Model class *Attribute* for a *Many-to-Many association*, *One-to-Many join table*, *Many-to-One join table*, or a *One to One - No Join Table* scenario, make sure to supply the same value for both the *implements-association* and *inverse-of* Tag Values of the bi-directional association. |   See also the related *implements-association* Tag Value. |
| lazy-load | A Tag Value added to an *Association* element between two Logical Model classes. The value specifies whether the association should be fetched lazily or not.  Permissible values are *yes*, and *no*. That is, any value other than *yes* is treated as a *no*. Sets the lazy attribute in the generated *.hbm.xml* file to either *true* or *false* accordingly.  No example is provided in the SDK sample model. |
| mapped-attributes | A Tag Value added to a Data Model class *Attribute* (Column). The value specifies the corresponding mapped Logical Model class *Attribute*. The value must be specified using the following pattern: *<fully-qualified logical model class name>.< attribute name>.*  A couple of examples from the SDK sample model are provided in the table below:   |  |  |  | | --- | --- | --- | | Data Model Class (Table) | Data Model Attribute (column) | Tag Value (mapped-attributes) | | UNDERGRADUATE\_ STUDENT | STUDENT\_ID | gov.nih.nci.cacoresdk.domain.inheritance.multiplechild.UndergraduateStudent.id | | SHOES | ID | gov.nih.nci.cacoresdk.domain.inheritance.childwithassociation.sametable.Shoes.id | |
| mapped-collection-table | A Tag Value added to a Logical Model class *Attribute*. The value specifies the name of the mapped primitive collection (non-domain class – e.g., String, Integer) table.  Given the following Data Model class diagram from the SDK sample model:    An example *mapped-collection-table* Tag Value would be:   |  |  |  | | --- | --- | --- | | Logical Model Class | Logical Model Attribute | Tag Value (mapped-collection-table) | | AllDataType | stringCollection | ALL\_DATA\_TYPE\_STRING\_COLL | |
| mapped-element | A Tag Value added to a Data Model class *Attribute* (Column). The value specifies the name of the mapped primitive collection (non-domain class – e.g., String, Integer) Logical Model class attribute. The value must be specified using the following pattern: *<fully-qualified logical model class name>.< attribute name>.*  Given the following Logical Model class diagram:    And the following Data Model class diagram from the SDK sample model:    An example *mapped-collection-table* Tag Value would be:   |  |  |  | | --- | --- | --- | | Data Model Class (Table) | Data Model Attribute (Column) | Tag Value (mapped-element) | | ALL\_DATA\_ TYPE\_STRING\_COLL | STRING\_VALUE | gov.nih.nci.cacoresdk.domain.other.datatype.AllDataType.stringCollection | |
| type | A Tag Value added to a Data Model class *Attribute* (Column). The value specifies the DB column type. Valid values include (but are not limited to):   * CHAR * CLOB * NUMBER * VARCHAR2   Several examples from SDK sample model include:   |  |  |  | | --- | --- | --- | | Data Model Class (Table) | Data Model Attribute (Column) | Tag Value (mapped-element) | | CHARACTER\_ PRIMITIVE\_KEY | ID | CHAR | | CARD | IMAGE | CLOB | | UNDERGRADUATE\_STUDENT | STUDENT\_ID | NUMBER | | SHOES | COLOR | VARCHAR2 | |

## Exporting the UML Model to XMI (EA Only)

|  |  |
| --- | --- |
| NOTE: | This section only applies to EA as the ArgoUML project is stored in an XML format that the SDK Code Generator can understand and process directly; i.e., the ArgoUML project file does *not* need to be exporting to XMI prior to processing it via the SDK Code Generator. |

Before the SDK can process a UML model created within Enterprise Architect (EA), it needs to be exported to XMI and then copied to the *\models* directory within the SDK root folder.

To export a package to XMI, follow these steps:

1. In the EA *Project Browser*, select the Logical View package, as shown below:

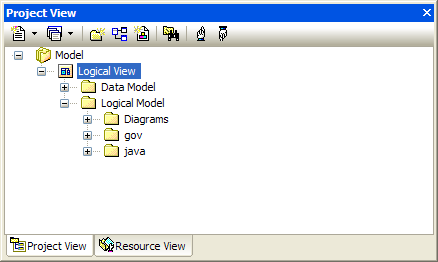


Figure EA Logical View Package

1. Right click and select the *Import/Export* menu OR go to the *Project | Import/Export* submenu. Select the *Export Package to XMI* option.
2. The Export Package to XMI dialog will appear, as shown below:

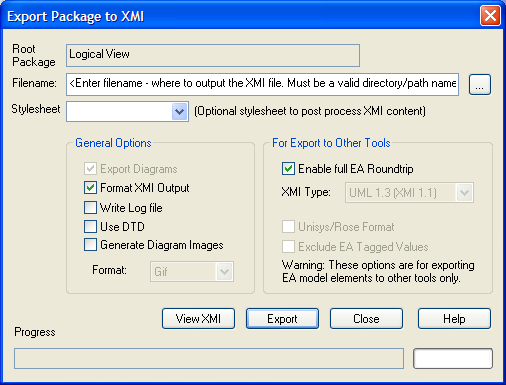


Figure EA Exporting Package to XMI

1. Set the *Export* options as follows:

|  |  |
| --- | --- |
| NOTE: | The XMI export options that should be selected have changed for SDK 4.0. The new required options are:   * Export Diagrams * Enable full EA Roundtrip |

| **Export Option** | **Description** |
| --- | --- |
| Filename | Used to indicate where to output the XMI file. *Enter a valid directory/path name. Also, make sure the file type suffix is .xmi.* |
| Stylesheet | Used to post-process XMI content before saving to file. *Leave unselected.* |
| Export Diagrams | *Leave checked.* |
| Use Unisys Rose Format | Used to indicate whether or not the Model should be exported in Rose UML 1.3, XMI 1.1 format. *Leave unchecked.* |
| Format XML output | Used to indicate whether or not to format output into readable XML (takes a few additional seconds at end of run). *Leave checked.* |
| Write log file | Used to indicate whether or not a log of export activity should be created (recommended). The log file will be saved in the same directory exported to. *Optional. Leave checked if desired.* |
| Use DTD | Used to indicate whether or not to use the UML1.3 DTD. Using this option will validate the correctness of the model and that no syntactical errors have occurred. *Leave unchecked.* |
| Exclude EA Tagged Values | Used to indicate whether or not EA specific information should be excluded from the export to other tools. *The SDK now supports Full EA roundtrip. Leave unchecked.* |

1. Press *Export*.
2. Once the XMI file has been exported, copy it to the *\models* directory within the SDK root folder.

|  |  |
| --- | --- |
| NOTE: | The XMI file name and the value of the *MODEL\_FILE* property within the *deploy.properties* file must match. Otherwise, a *File Not Found* error will be reported when trying to process the XMI file through the SDK Code Generator. |

## Importing XMI into the UML Model (EA Only)

|  |  |
| --- | --- |
| NOTE: | This section only applies to EA as the ArgoUML project is stored in an XML format that the SDK Code Generator can understand and process directly; i.e., the ArgoUML project file does *not* need to be exporting to XMI prior to processing it via the SDK Code Generator. |

The SDK now supports the processing of XMI that was exported using the full EA roundtrip option. Some organizations may have the need to modify the exported XMI file, perhaps to add Tag Values. As long as the XMI was exported using the *roundtrip* option, it can be synchronized with the UML model by importing it back into EA.

|  |  |
| --- | --- |
| NOTE: | The selection of the incorrect import options may corrupt the model file.  *Make sure to backup the original model file prior to importing XMI back into the UML model!!!* |

To import an XMI package back into EA, follow these steps:

1. In the EA *Project Browser*, select the Logical View package, as shown below:

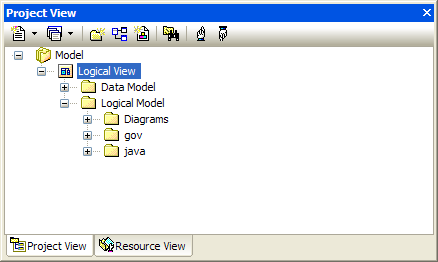


Figure EA Logical View Package

1. Right click and select the *Import/Export* menu OR go to the *Project | Import/Export* submenu. Select the *Import Package from XMI* option.
2. The Import Package from XMI dialog will appear, as shown below:

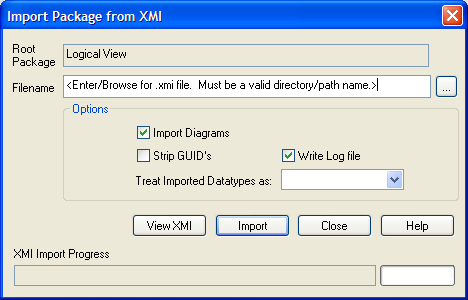


Figure EA Import Package from XMI

1. Set the *Import* options as follows:

| **Import Option** | **Description** |
| --- | --- |
| Filename | Used to indicate where to import the XMI file. *Enter a valid directory/path name.* |
| Import Diagrams | *Make sure to leave checked!!!* |
| Strip GUIDS | Used to remove Universal Identifier information from the file on import. This permits the import of a package twice into the same model - the second import will require new GUIDS to avoid element collisions. *Make sure to leave unchecked!!!* |
| Treat Imported Datatypes as | *Leave unselected.* |
| Write log file | Used to indicate whether or not a log of export activity should be created (recommended). The log file will be saved in the same directory exported to. *Optional. Leave checked if desired.* |

1. Click *Import*.
2. A confirmation dialog will appear, as shown below:

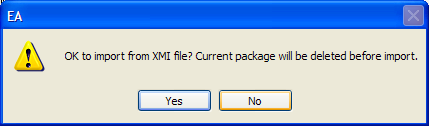


Figure EA Confirm XMI File Import Dialog

1. Click *Yes*. The XMI file will now be imported back into EA, and the XMI and UML model will be synchronized.

1. See <http://en.wikipedia.org/wiki/Model-driven_architecture> for more information on Model-Driven Architecture software design approach. [↑](#footnote-ref-2)
2. See <http://argouml-stats.tigris.org/documentation/manual-0.24/ch11.html> for more information on the ArgoUML Explorer pane. [↑](#footnote-ref-3)
3. See <http://argouml-stats.tigris.org/documentation/manual-0.24/ch13s03.html> for more information on the ArgoUML Detail Pane, Properties tab. [↑](#footnote-ref-4)
4. See <http://argouml-stats.tigris.org/documentation/manual-0.24/ch12.html> for more information about the Editing pane. [↑](#footnote-ref-5)
5. See <http://argouml-stats.tigris.org/documentation/manual-0.24/ch12.html> for more information about the Editing pane. [↑](#footnote-ref-6)
6. For more information regarding the caAdapter tool/project, see <http://trials.nci.nih.gov/projects/infrastructureProject/caAdapter>. [↑](#footnote-ref-7)
7. The <discriminator> element is required for polymorphic persistence using the table-per-class-hierarchy mapping strategy and declares a discriminator column of the table. The discriminator column contains marker values that tell the persistence layer what subclass to instantiate for a particular row. See <http://www.hibernate.org/hib_docs/v3/reference/en/html_single/#mapping-declaration-discriminator> for more information. [↑](#footnote-ref-8)