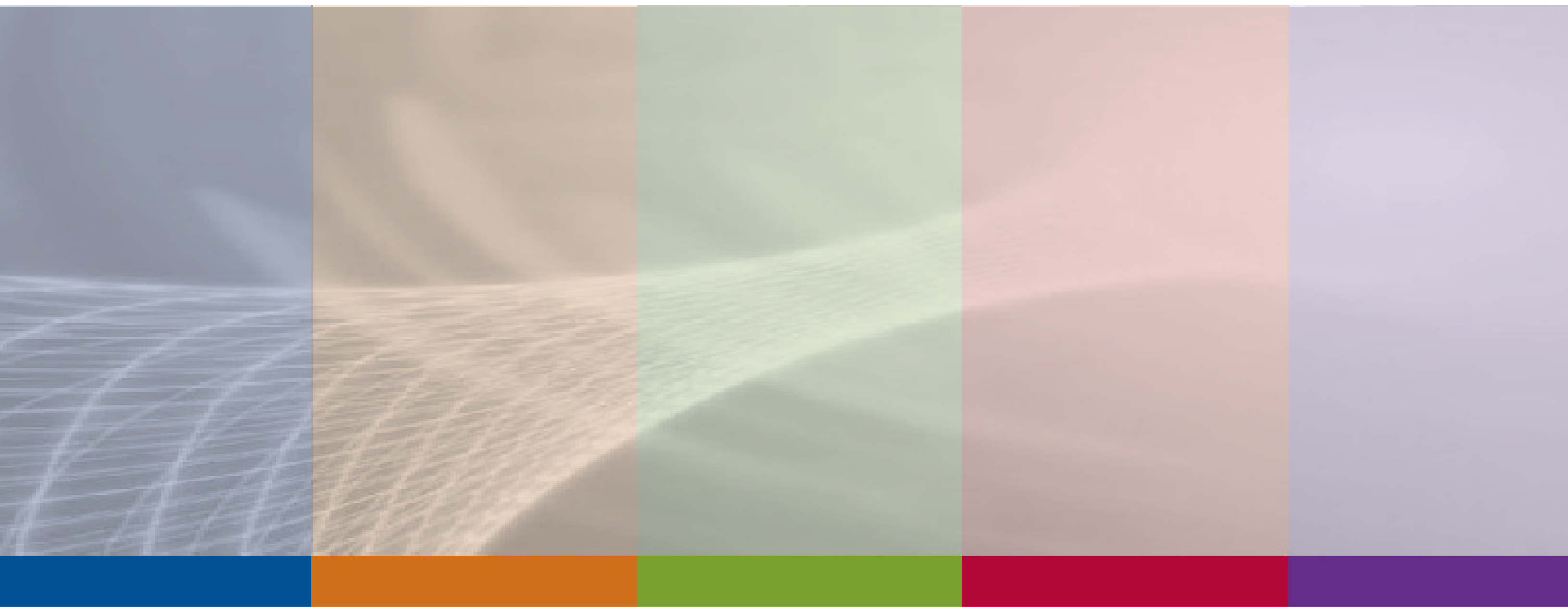
metamatrix

MetaBase Modeler User's Guide

Release 4.2 SP2

June 2005



MetaBase Modeler User's Guide
MetaMatrix Products, Release 4.2 SP2 (Second Service Pack for Release 4.2)
Document Edition 1, June 10, 2005

If you have any questions, comments, or suggestions regarding this document, contact documentation@metamatrix.com.

Table of Contents

Chapter 1: Modeling Data

THE BUSINESS CHALLENGE

Many organizations have come to depend on numerous sources of information for their daily operation. Different departments and divisions within a company might have developed their own information technology solutions. Two companies with different IT philosophies may have merged. There are many reasons why a business may find itself struggling to tie together disparate information sources, such as Relational databases, inventory management systems, and/or legacy systems. As businesses become interconnected through the Internet, networking technologies, and partnerships, there are more sources of information available to an enterprise.

Your organization faces the challenge of using all of its information sources to their full potential. Attempting to catalog exactly what information lies within a jumble of enterprise information systems can clutter many white boards and notebooks.

In many cases, it makes perfect sense for different areas of your enterprise to use different applications to access multiple individual enterprise information systems. But as the number of enterprise information systems in your business increases, the number of the applications you use to get information can increase as well. This web of applications can decrease your day-to-day business operation efficiency as you navigate multiple information sources.

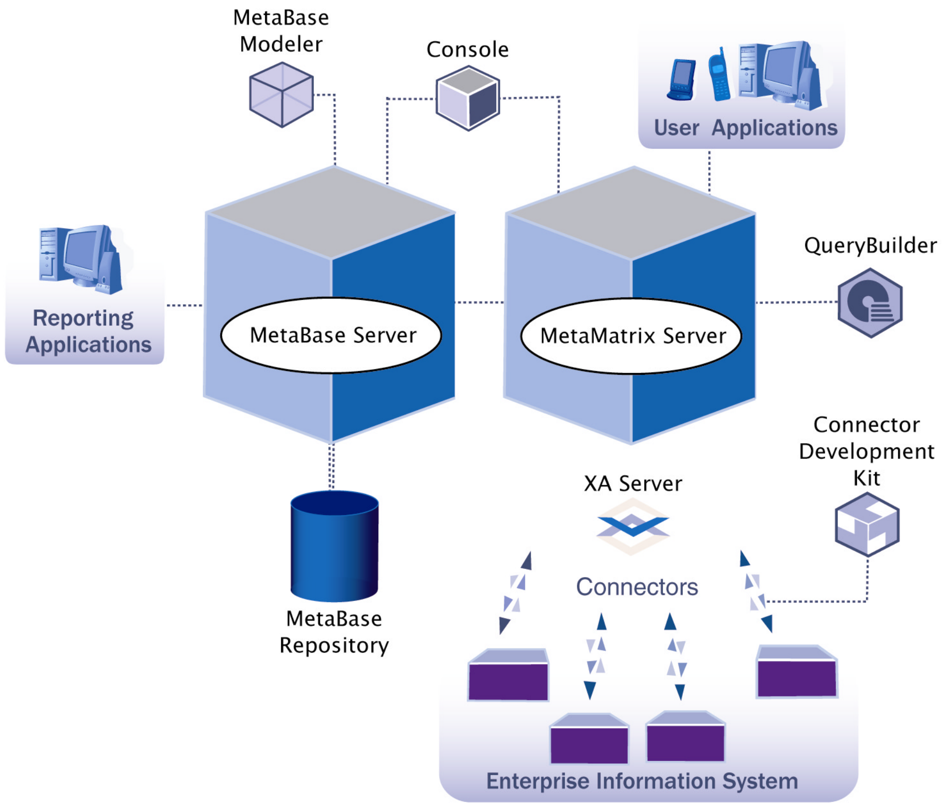
The challenge that faces your organization is organizing the complex and often interrelated sources of information needed to compete and remain efficient.

THE METAMATRIX SOLUTION

The MetaMatrix System (comprised of the MetaMatrix Server and the MetaBase metadata management system) offers your organization a way to manage and describe the information across your disparate enterprise information systems. You can even integrate these enterprise information systems into a single, complete data access solution using the MetaMatrix Server.

The MetaMatrix System

The entire MetaMatrix System is comprised of several interconnected products and services:



The **MetaMatrix System**, when used in its totality, enables your end user applications to process queries that select (and even update) data from one or more of your enterprise information sources, regardless of the native physical data storage method used by each enterprise information system. This means that a *single* query can access, reference, and return results from multiple integrated data sources.

Within the MetaMatrix System, the MetaBase products (including the **MetaBase Modeler**, the **MetaBase Server**, and the **MetaBase Repository)**, enable you to create and manage metadata models: representations describing the nature and content of your enterprise information systems.

Once captured, this valuable metadata can be searched, analyzed, and applied by applications throughout your enterprise.

These metamodels can be deployed to the MetaMatrix Server. The server can use the metadata at runtime to:

- Process queries posed by the user application of your choice
- Retrieve from the information source(s) of your choice
- Return the integrated results in the information format of your choice

The MetaMatrix Server parses queries based upon the metadata information and distributes the subqueries to the appropriate enterprise information system(s) through **Connectors**. These connectors are Java classes that translate queries into the enterprise information system's native application programming interface (API). Once the various enterprise information systems return the data results, the MetaMatrix Server reassembles and returns those results to the client application of your choice.

Chapter 2: What is Metadata?

Metadata is data about data. A piece of metadata, called a meta object in the MetaMatrix MetaBase Modeler, contains information about a specific information *structure*, irrespective of whatever individual data fields that may comprise that structure.

Let's use the example of a very basic database, an address book. Within your address book you certainly have a field or column for the ZIP code (or postal code number). Assuming that the address book services addresses within the United States, you can surmise the following about the column or field for the ZIP code:

- Named ZIPCode
- Numeric
- A string
- Nine characters long
- Located in the StreetAddress table.
- Comprised of two parts: The first five digits represent the five ZIP code numbers, the final four represent the ZIP Plus Four digits if available, or 0000 if not.
- Formatted only in integer numeric characters. Errors will result if formatted as 631410.00 or 6314q0000.

This definition represents metadata about the ZIP code data in the address book database. It abstracts information from the database itself and becomes useful to describe the content of your enterprise information systems and to determine how a column in one enterprise information source relates to another, and how those two columns could be used together for a new purpose.

You can think of this metadata in several contexts:

- What information does the metadata contain? For more information, see "Business and Technical Metadata."
- What data does the metadata represent? For more information, see "Physical and Virtual Metadata."
- How will my organization use and manage this metadata? For more information, see "Design-Time and Runtime Metadata."

EDITING METADATA VS. EDITING DATA

The MetaBase Modeler helps you to create a graphic representation of your data. This abstracted, graphic representation defines and describes the structure and layout of your data in the original data sources. It also describes whether those data sources are composed of Relational databases, text files, data streams, legacy database systems, or some other information type.

The MetaBase Modeler creates, edits, and links these graphically-represented meta objects that are really a *description* of your data, and not the data itself. So when this documentation describes the process of creating, deleting, or editing these meta objects, remember that you are not, in fact, modifying the underlying data.

METADATA MODELS

A metadata model represents a collection of metadata information that describes a complete structure of data.

In a previous example we described the field ZIPCode as a metadata object in an address book database.. This meta object represents a single distinct bit of metadata information. We alluded to its parent table, StreetAddress. These meta objects, and others that would describe the other tables and columns within the database, would all combine to form a physical metadata model for whichever enterprise information system hosts all the objects.

You can have **physical models** within your collection of metadata models. These model physical data storage locations. You can also have **virtual models**, which model the business view of the data. Each contains one type of metadata or another. For more information about difference between physical and virtual metadata, see "Physical and Virtual Metadata."

NOTE: *For more information about using models as you model your metadata, see "Modeling Your Metadata."*

BUSINESS AND TECHNICAL METADATA

Metadata can include different types of information about a piece of data.

- **Technical metadata** describes the information required to access the data, such as where the data resides or the structure of the data in its native environment.

- **Business metadata** details other information about the data, such as keywords related to the meta object or notes about the meta object.

Note that the terms "technical" and "business" metadata refer to the content of the metadata, namely what type of information is *contained* in the metadata. Don't confuse these with the terms "physical" and "virtual" metadata that indicate what the metadata *represents*. For more information, see "Physical and Virtual Metadata."

Technical Metadata

Technical metadata represents information that describes how to access the data in its original native data storage. Technical metadata includes things such as datatype, the name of the data in the enterprise information system, and other information that describes the way the native enterprise information system identifies the meta object.

Using our example of an address book database, the following represent the technical metadata we know about the ZIP code column:

- Named ZIPCode.
- Nine characters long.
- A string.
- Located in the StreetAddress table.
- Uses SQL Query Language

These bits of information describe the data and information required to access and process the data in the enterprise information system.

Business Metadata

Business metadata represents additional information about a piece of data, not necessarily related to its physical storage in the enterprise information system or data access requirements. Business metadata can represent descriptions, business rules, and other additional information about a piece of data.

Continuing with our example of the ZIP Code column in the address book database, the following represents business metadata we may know about the ZIP code:

- The first five characters represent the five ZIP code numbers, the final four represent the ZIP Plus Four digits if available, or 0000 if not.
- The application used to populate this field in the database strictly enforces the integrity of the data format.

Although the first might seem technical, it does not directly relate to the physical storage of the data. It represents a business rule applied to the contents of the column, not the contents themselves.

The second, of course, represents some business information about the way the column was populated. This information, although useful to associate with our definition of the column, does not reflect the physical storage of the data.

PHYSICAL AND VIRTUAL METADATA

In addition to the distinction between business and technical metadata, you should know the difference between *physical* metadata and *virtual* metadata. Physical and virtual metadata refer to what the metadata represents, not its content.

Physical metadata directly represents metadata for an enterprise information system and captures exactly where and how the data is maintained. Physical metadata sounds similar to technical metadata, but physical metadata can contain *both* technical and business metadata. When you model physical metadata, you are modeling the data that your enterprise information systems contain. For more information, see "Modeling Your Enterprise Information Systems."

Virtual metadata, on the other hand, can create one or more tailored views that transform the physical metadata into the terminology and domain of different applications. Virtual metadata, too, can contain both technical and business metadata. When you model virtual metadata, you're modeling the data as your applications (and your enterprise) ultimately use it. For more information, see "Modeling Your Enterprise Data Needs."

What Is An Enterprise Information System?

The term enterprise information system (EIS) represents a physical source of data that your enterprise uses in its business activity. Your enterprise probably derives information from numerous sources, including Relational database management systems (RDBMS), streaming Internet data feeds, text files, legacy systems, and others.

Modeling Your Enterprise Information Systems

When you model the physical metadata within your enterprise information systems, you capture some detailed information, including:

- Identification of datatypes
- Storage formats
- Constraints
- Source-specific locations and names

The physical metadata captures this detailed technical metadata to provide a map of the data, the location of the data, and how you access it.

This collection of physical metadata comprises a direct mapping of the information sources within your enterprise. If you use the MetaMatrix Server for information integration, this technical metadata plays an integral part in query resolution.

For example, our ZIPCode column and its parent table StreetAddress map directly to fields within our hypothetical address book database.

To extend our example, we might have a second source of information, a comma-separated text file provided by a marketing research vendor. This text file can supply additional demographic information based upon address or ZIP code. This text file would represent another EIS, and the meta objects in its physical model would describe each comma-separated value.

Modeling Your Enterprise Data Needs

When you create virtual metadata, you are not describing the nature of your physical data storage. Instead, you describe the way your enterprise uses the information in its day-to-day operations.

Virtual metadata derives its classes and attributes from other metadata. You can derive virtual metadata from physical metadata that describes the ultimate sources for the metadata or even from other virtual metadata. However, when you model virtual metadata, you create special "views" on your existing enterprise information systems that you can tailor to your business use or application expectations. This virtual metadata offers many benefits:

- You can expose only the information relevant to an application. The application uses this virtual metadata to resolve its queries to the ultimate physical data storage.
- You can add content to existing applications that require different views of the data by adding the virtual metadata to the existing virtual metadata that application uses. You save time and effort since you do not have to create new models nor modify your existing applications.
- Your applications do not need to refer to specific physical enterprise information systems, offering flexibility and interchangeability. As you change sources for information, you do not have to change your end applications.
- The virtual metadata models document, in a central place, the various ways your enterprise uses the information and the different terminology that refers to that information.

Our example enterprise information sources, the address book database, and the vendor-supplied comma-delimited text file, reside in two different native storage formats and therefore have two physical metadata models. However, they can represent one business need: a pool of addresses for a mass mailing.

By creating a virtual metadata model, we could accurately show that this single virtual table, the AddressPool, contains information from the two enterprise information systems. The virtual metadata model not only shows from where it gets the information, but also the SQL operations it performs to select its information from its source models.

This virtual metadata can not only reflect and describe how your organization uses that information, but, if your enterprise uses the MetaMatrix Server, your applications can use the virtual metadata to resolve queries.

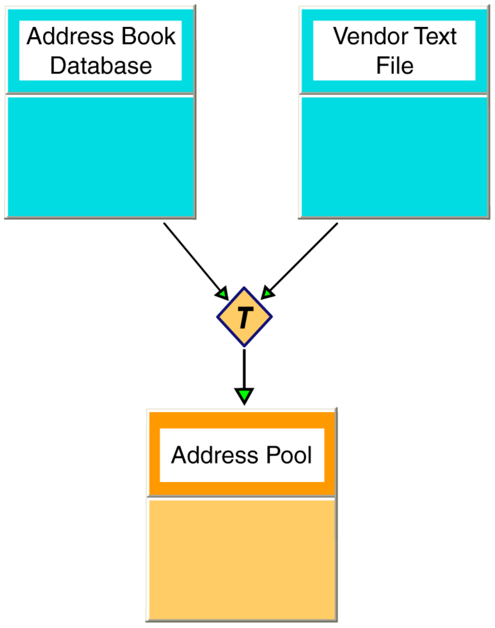
To create this virtual metadata, you create a **transformation**, a special query that enables you to select information from the physical (or even other virtual) metadata models. For more information, see "Modeling Transformations."

MODELING METADATA TRANSFORMATIONS

Metadata Transformations

By modeling virtual metadata, you can illustrate the business view of your enterprise information sources. Virtual metadata models not only describe that business view, but also illustrate how the meta objects within the virtual metadata models derive their information from other metadata models.

Let's return to the example of our address book database and the vendor's comma-separated list. We want to generate the virtual metadata model, Address Pool, from these enterprise information systems.



The transformation that joins these metadata models to create the virtual Address Pool metadata model contains a SQL query, called a **union**, that determines what information to draw from the source metadata and what to do with it. For more information about the SQL you can include in your transformations, see "SQL in Transformations."

The resulting Address Pool contains not only the address information from our Address Book database, but also that from our vendor-supplied text file.

SQL in Transformations

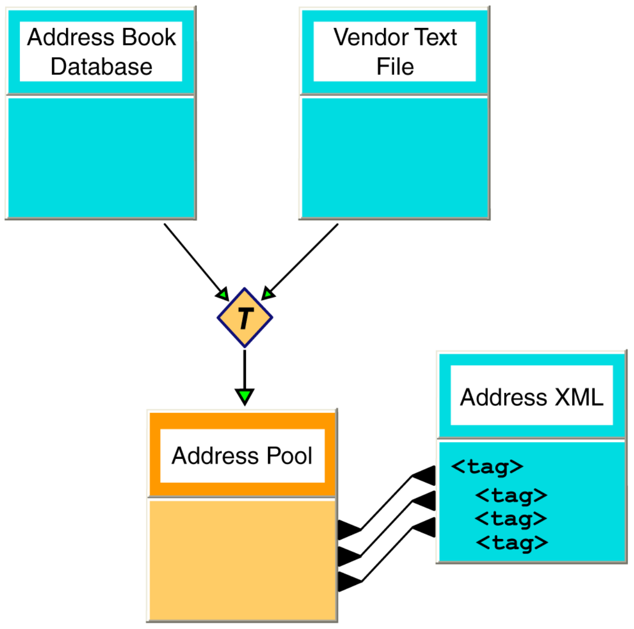
Transformations contain SQL queries that SELECT the appropriate attributes from the information sources.

For example, from the sources the transformation could select relevant address columns, including first name, last name, street address, city, state, and ZIP code. Although the metadata models could contain other columns and tables, such as phone number, fax number, e-mail address, and Web URL, the transformation acts as a filter and populates the Address Pool metadata model with only the data essential to building our Address Pool.

You can add other SQL logic to the transformation query to transform the data information. For example, the address book database uses a nine-character string that represents the ZIP Plus Four. The transformation could perform any SQL-supported logic upon the ZIPCode column to substring this information into the format we want for the Address Pool virtual metadata model.

Mapping XML Transformations

When you model virtual metadata, you can create a virtual XML document model. This virtual document lets you select information from within your other data sources, just like a regular virtual metadata model, but you can also map the results to tags within an XML document.



In this example, the Address Pool virtual metadata model still selects its information from the Address Book Database and the Vendor Text File, but it also maps the resulting columns into tags in the Address XML document.

DESIGN-TIME AND RUNTIME METADATA

MetaMatrix software distinguishes between **design-time** metadata and **run-time** metadata. This distinction becomes important if you use both MetaMatrix MetaBase *and* the MetaMatrix Server. Design-time data is laden with details and representations that help the user understand and efficiently organize metadata. Much of that detail is unnecessary to the underlying system that runs the Virtual Database that you will create. Any information that is not absolutely necessary to running the Virtual Database is stripped out of the run-time metadata to ensure maximum system performance.

Design-Time Metadata

Design-time metadata refers to data within the MetaBase Repository or your local directory that you have created or have imported. You can model this metadata in the MetaBase Modeler, adding physical and virtual metadata. If you only use the MetaBase product, you will work exclusively with design-time metadata.

The MetaBase Modeler handles design-time metadata, but within the MetaBase Repository Manager you take the preliminary steps to create the runtime metadata. For more information, see "Sharing Models and Projects in the Team Repository."

Runtime Metadata

Once you have adequately modeled your enterprise information systems, including the necessary technical metadata that describes the physical structure of your EISes, you can use the metadata for data access.

To prepare the metadata for use in the MetaMatrix Server, you take a snapshot of a metadata model for the MetaMatrix Server to use when resolving queries from your client applications. This run-time metadata represents a static version of design-time metadata you created or imported. As you create this runtime metadata, the MetaBase Modeler:

- Derives the runtime metadata from a consistent set of metadata models.
- Creates a subset of design-time metadata, focusing on the technical metadata that describes the access to underlying enterprise information systems.
- Optimizes runtime metadata for data access performance.

You can continue to work with the design-time metadata, but once you have created a runtime metadata model, it remains static.

Chapter 3: Metamodels in the MetaBase Modeler

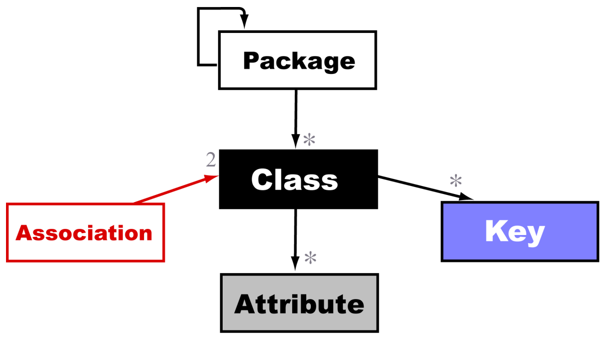
WHAT IS A METAMODEL?

Metamodels define which properties, constructs, and terminology are available to describe information. The type of information you can capture in a metadata model comes from the metamodel. For example, our metadata representation of our ZIPCode column has a length associated with it because the metamodel it uses contains a construct called a column which has a property called length.

The MetaBase Modeler uses metamodels to capture metadata according to the Object Management Group's Meta Object Facility standard.

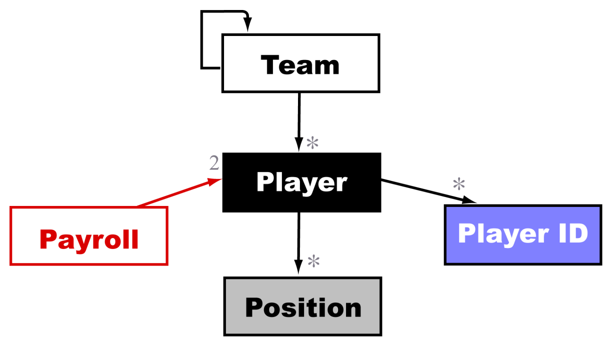
NOTE: *For more information about this standard, see the* Meta object Facility (MOF) Specification, Version 1.3, March 2000, *available from the Object Management Group.* http://www.omg.org

This aforementioned standard describes metamodels like this:



Each metadata model created with a metamodel can have the following components, or meta objects, within it:

- **Package**, which can contain one or more instances of a class or package.
- **Class**, which can contain one or more attributes and keys.
- **Attribute**, one or more of which belong to an instance of the class.
- **Key**, one or more of which belong to an instance of the class.
- **Associations**, which can exist between classes.



The MetaMatrix MetaBase Modeler supports several different metamodels that adhere to this standard. For more information, see "Metamodels in the MetaBase Modeler."

Throughout this document, you'll find the terms **package**, **class**, and **attribute** refer to the different meta objects allowed in a metamodel.

METAMODELS IN THE METABASE MODELER

In earlier releases, the MetaBase Modeler supported only one metamodel, the Data Access metamodel. This metamodel provided a single terminology to refer to the meta objects and the categories, groups, and elements that comprised your metadata model.

However, the MetaBase Modeler now supports different metamodels. These metamodels reflect the way your different enterprise information systems organize the information within and the different terminology associated with each.

For example, you can select the Relational metamodel when modeling or importing metadata from a Relational database management system, but this changes the nomenclature available on the menus and displays to reflect that specific metamodel's constructs and terminology.

METAMODEL EXTENSIBILITY

The MetaMatrix MetaBase Modeler provides a great deal of extensibility in data modeling because the MetaBase Modeler recognizes more than one metamodel. Therefore, you can more accurately maintain the names and domain structure of your enterprise information systems (EISes) by modeling them in the terminology native to that type of enterprise information system. You can even create new metamodels that uniquely capture metadata to describe practically any enterprise information system.

The metamodels in the MetaMatrix MetaBase Modeler are data-driven. This means all rules inherent in the metamodel that the MetaBase Modeler enforces regarding the packages, classes, and attributes are not hard-coded in the MetaBase Modeler. All properties available for the packages, classes, and attributes come from the metamodel as well.

Future versions of the MetaMatrix MetaBase Modeler will include other metamodels, and because of this extensibility, you can create models using the new metamodels and seamlessly integrate them, including virtual metadata created through transformations, with your existing models.

AVAILABLE METAMODEL TYPES

The MetaMatrix MetaBase Modeler Release 4.2 currently supports the following metamodels:

- **Relational**, which contains packages, classes, and attributes commonly used by Relational databases. For more information, see "The Relational Metamodel."
- **Data Access**, which contains package, class, and attribute terminology used by the MetaMatrix Server in 1.x releases to resolve queries. For more information, see "The Data Access Metamodel."
- **XML Documents,** used to capture the structure of a virtual document.
- **Relational,** used to capture the structure (system catalog) of relational sources.
- **Relationship,** a secondary metamodel that provides generalized relationships between any model objects and provides a mechanism to constrain which model objects can participate in the relationships of the corresponding type.
- **People,** a secondary metamodel that provides a simple model of people, typically used in conjunction with generalized relationships.
- **UML2,** provides the subset of UML 2.0 defining static and structural modeling capabilities.
- **Transformation,** used to capture the structure and detailed information about a transformation from one or more source classifiers.
- **Diagram,** a secondary metamodel used to capture the information about a diagram.
- **History,** a secondary metamodel used to represent the history of an item in the repository.
- **Dependency,** a secondary metamodel used to capture and represent the dependencies a model has on other models.
- **Core,** a secondary metamodel core set of metamodel constructs, including those used to annotate model objects with a description.
- **Model Extension,** a metamodel that defines extensions to other metamodels.
- **Model Compare,** a metamodel used to represent and persist model comparisons.
- **JDBC,** used to represent JDBC driver libraries and JDBC sources.
- **Function,** used to capture the user-defined functions that are available to the MetaMatrix Server. Note that the category property is a **required** property in this metamodel.
- **VDB,** used in the manifest model of a virtual database file to capture the models that make up the virtual database and other information about the information contained within the virtual database file.
- **DQP,** used in the configuration model for an embedded DQP component.

- **Web Services,** a model builder for generating a Web Services model from a Web Services Definition (WSDL) file.

Not every possible setting for every possible metamodel is covered in this manual. Some of the different metamodel terms include:

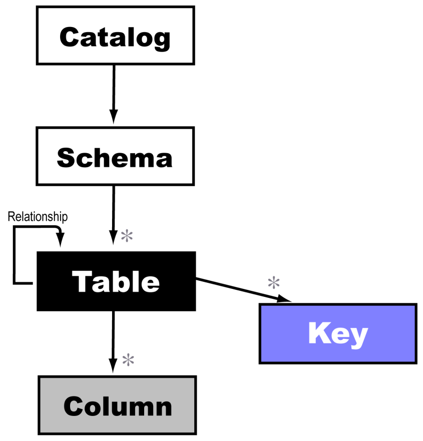
Metamodel	Package	Class	Attribute
Data Access	Category	Group	Element
Relational	Schema, Catalog	Table, View	Column, Key
Simple Datatypes	Domain	Atomic Simple Type List Simple Type Union Simple Type	Facet
XML Document	Document	Element, Attribute	*None*
Web Services	Interface	Operation	Input, Output

In other MetaMatrix documentation, this distinction does not exist; once you create a virtual database (VDB) for use with the MetaMatrix Server, the VDB uses the Data Access metamodel. Therefore, other documentation uses the Category/Group/Element type terminology to refer to the Package, Class, and Attribute.

These metamodels are based upon the Object Management Group's Meta Object Facility (MOF) and Common Warehouse Metamodel (CWM) standards.

The Relational Metamodel

The Relational metamodel describes metadata (and so the native data storage) in terms associated with Relational database management systems (RDBMS). The Relational metamodel reflects the following organization of information:

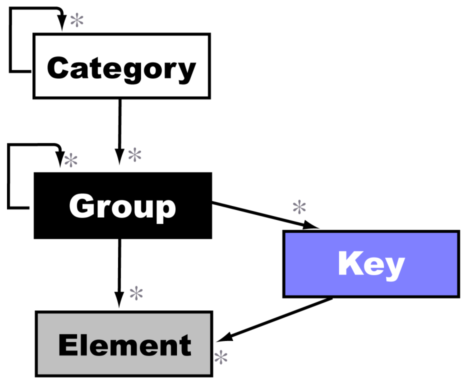


As such, this metamodel names the Packages, Classes, and Attributes as follows:

Package	Class	Attribute
Schema, Catalog	Table, View, Result Set, Stored Procedure	Column, Key

The Data Access Metamodel

The Data Access metamodel describes metadata (and the native data storage) in terminology used by the MetaMatrix Server in 1.x releases. The Data Access metamodel reflects the following organization of information:

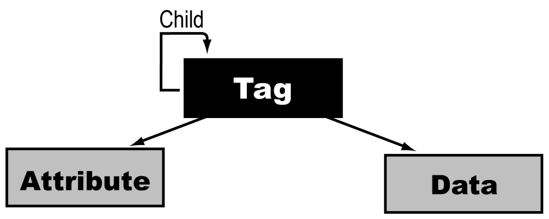


As such, this metamodel names the Packages, Classes, and Attributes as follows:

Package	Class	Attribute
Category	Group	Element

The XML Metamodels

The metamodels for XML Schema, and XML Documents reflect the structure, including tags and attributes, within XML files.



The XML Schema metamodel enables you to model the constraints within an XML Schema; the XML Document metamodel enables you to describe the contents of an actual XML instance document.

Choosing a Metamodel

When you begin to model your enterprise information systems, one of the first decisions you must make is which metamodel you should use to represent your physical or virtual metadata. You should base this decision by determining the constructs and the terminology you want in your metadata model.

The **Relational metamodel** contains terms and constructs specific to Relational databases. The constructs, which not only include the name of the packages, classes, and attributes but how they relate to one another, reflect those common to Relational databases. In general, you want to create a physical metadata model of a Relational database using the Relational metamodel. The Relational metamodel also contains constructs that others do not, such as stored procedures and result sets.

The **Data Access metamodel** contains more generic constructs and terms by design. You can apply the constructs within it not only to Relational databases, but other types of enterprise information systems. Therefore you can use the Data Access metamodel to model any sort of physical enterprise information system.

The **XML metamodel** let you create models that describe the XML documents your organization uses to exchange information.

When creating virtual metadata models, you can choose a metamodel that suits your purpose for modeling the virtual metadata.

If you want to use constructs and terminology that emphasizes the relational nature of your data, such as information from Relational databases, you can use the Relational metamodel.

However, if you want to emphasize the abstraction, which is especially useful in more abstract virtual models that relate to actual physical data storage only through several transformations, you might choose the Data Access metamodel.

Chapter 4: Getting Started with the MetaBase Modeler

WHAT IS THE METABASE MODELER?

The MetaBase Modeler is an interface that enables you to capture, model, and maintain metadata for your organization's disparate enterprise information systems (EISes) and business views of those systems. You can model the form and structure of each EIS and logically name and organize the data independently of the physical data store. You can also import metadata from a variety of formats, including directly from some databases. Once you have created the metadata you need, you can store it as a metadata model, a set of related metadata sharing a common metamodel.

You can model this metadata to organize how your enterprise's information systems relate to one another. You can also create diagrams that illustrate the business rules your organization uses with the information within the enterprise information sources. MetaBase Modeler offers display and print capability for diagrams.

As you create models to represent the enterprise information systems your organization uses, you can use the MetaBase Modeler's MetaBase Repository Manager to store your models and enable other members of your organization to review or modify them.

Once you have organized your metadata into models, your organization, if you use the MetaMatrix Server, can then use those metadata models to perform queries using the data sources you have modeled.

The primary functions of the MetaBase Modeler include:

- Creating and editing metadata models. You can do this locally, without connecting to the MetaBase Server. For more information, see "Using the MetaBase Modeler Workspace."
- Sharing and storing metadata models. You connect to a MetaBase Server to store and version models in the MetaBase Repository. For more information, see "Sharing Models and Projects in the Team Repository."
- You can also use the MetaBase Modeler to define runtime metadata that the MetaMatrix Server uses to resolve queries. Every time a model is checked into the Team Share Repository, it is automatically added to the Design Time Catalog (DTC). The DTC can be searched using any standard reporting tool. For more information, see "Sharing Models and Projects in the Team Share Repository."

USING THE METABASE MODELER WORKSPACE

Modeling Metadata Locally

As you create your metadata models and populate those models with meta objects, diagrams, and transformations, you save your progress in your local directory. The MetaBase Modeler stores models using the Object Management Group (OMG) XML Metadata Interchange (XMI) standard.

The MetaBase Modeler handles this, the first of its primary functions, in the MetaBase Modeler window (also called the workspace). For more information about the MetaBase Modeler window, including its panel layout, see "Viewing the MetaBase Modeler Workspace Window."

Within this window, you can:

- Create metadata models and meta objects
- Import metadata from external sources, such as JDBC-compliant databases or XMI files
- Edit or view meta objects or diagrams
- Create transformations for virtual metadata

Note that you do not need a user name and password for the MetaMatrix System to model metadata within the MetaBase Modeler. You will, however, require a user name and password to connect to a MetaBase Server and its MetaBase Repository.

Opening the MetaBase Modeler Workspace Window

You can access the MetaBase Modeler Workspace in two ways.

If you're running the application, you can use the command line or Windows Start menu.

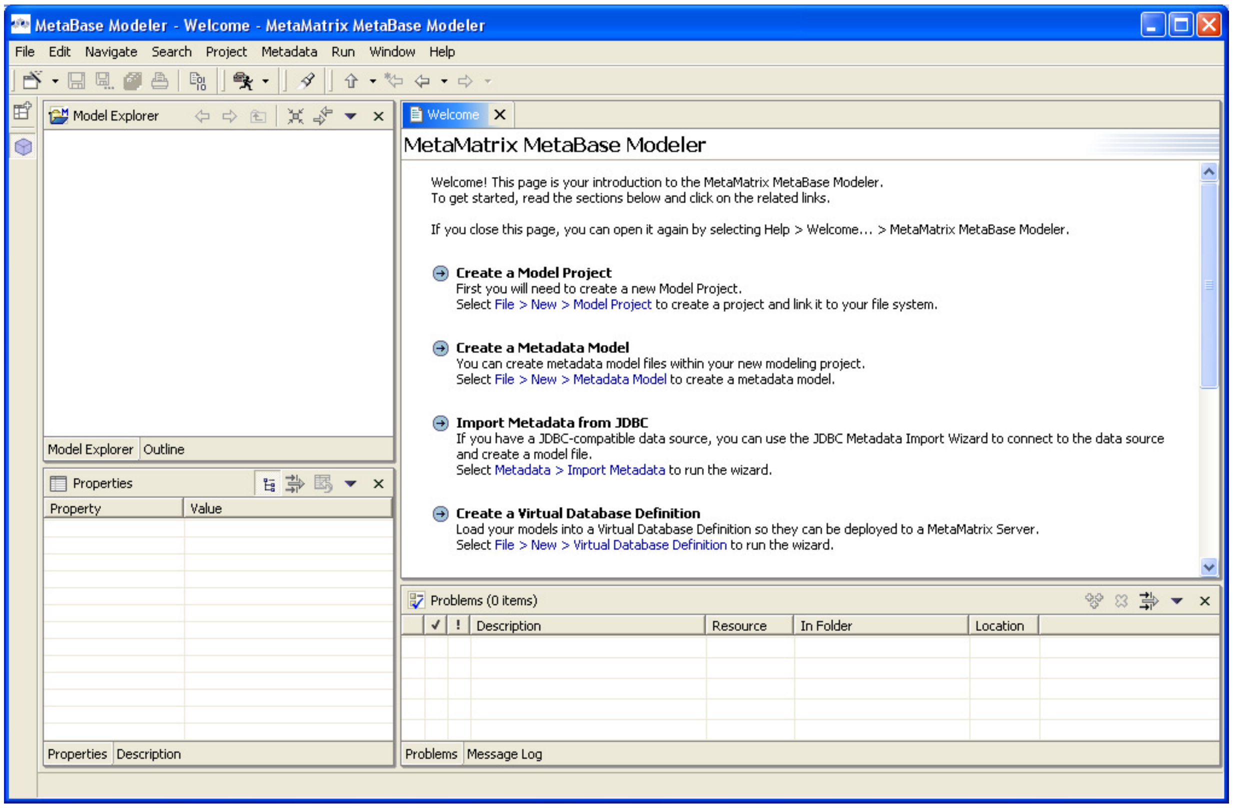
Running the Application

You can run the MetaBase Modeler by executing the script or batch file that runs the MetaBase Modeler. The name of this file depends upon the operating system of the workstation upon which you run the MetaBase Modeler.

In the Windows operating system, you select **Programs > MetaMatrix Modeler 4.2 > Modeler**.

Viewing the MetaBase Modeler Workspace Window

The MetaBase Modeler Workspace window looks like this the first time you open it:


MetaBase Modeler - Welcome - MetaMatrix MetaBase Modeler
File Edit Navigate Search Project Metadata Run Window Help
Model Explorer
Model Explorer Outline
Properties
Property Value
Properties Description
Welcome
Problems (0 items)
Description Resource In Folder Location
Problems Message Log

MetaMatrix MetaBase Modeler

Welcome! This page is your introduction to the MetaMatrix MetaBase Modeler.
To get started, read the sections below and click on the related links.

If you close this page, you can open it again by selecting Help > Welcome... > MetaMatrix MetaBase Modeler.

Create a Model Project
First you will need to create a new Model Project.
Select File > New > Model Project to create a project and link it to your file system.

Create a Metadata Model
You can create metadata model files within your new modeling project.
Select File > New > Metadata Model to create a metadata model.

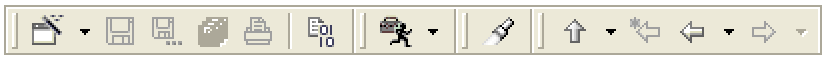
Import Metadata from JDBC
If you have a JDBC-compatible data source, you can use the JDBC Metadata Import Wizard to connect to the data source and create a model file.
Select Metadata > Import Metadata to run the wizard.

Create a Virtual Database Definition
Load your models into a Virtual Database Definition so they can be deployed to a MetaMatrix Server.
Select File > New > Virtual Database Definition to run the wizard.

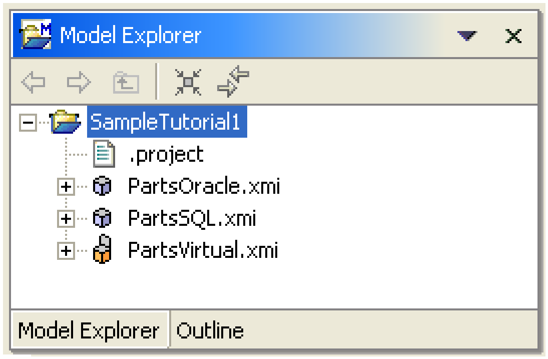
The window contains the following parts:

- The Menu bar, which contains a set of commonly used commands.

- The Main Toolbar, which provides one-click access to many common commands.



- The **Model Explorer** and **Outline** views, located in the upper left, provide a hierarchical view of your metadata. The **Model Explorer** view shows the contents of the entire project. Clicking the **Outline** tab toggles to show only the hierarchy of contents in the current **Editor Panel** view at any given time.

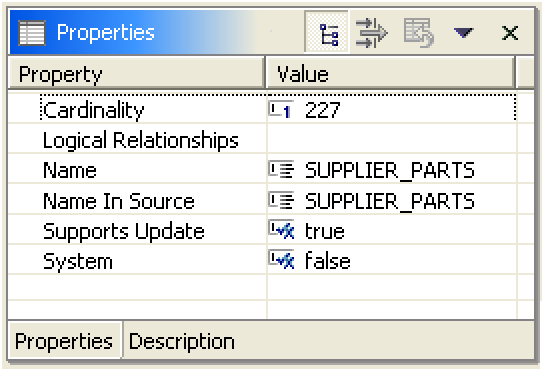


The **Model Explorer** view has some useful buttons to help you manage the contents of the view.

The **Collapse All** button closes all open nodes and reduces the view to its most basic profile.

The **Synchronize With Editor** button automatically changes the selection in **Model Explorer** view whenever an object is selected in the Editor Panel.

- The **Properties** and **Description** views offers detail about the meta object you have selected.

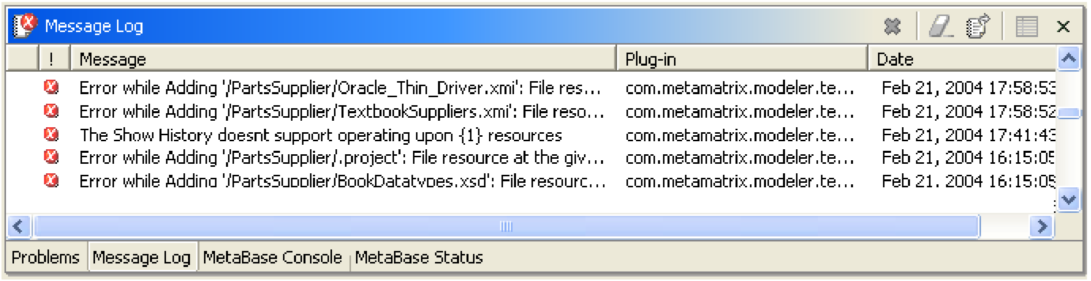


Property	Value
Cardinality	227
Logical Relationships	
Name	SUPPLIER_PARTS
Name In Source	SUPPLIER_PARTS
Supports Update	true
System	false

The **Properties** view has a useful button to help you manage the contents of the view.

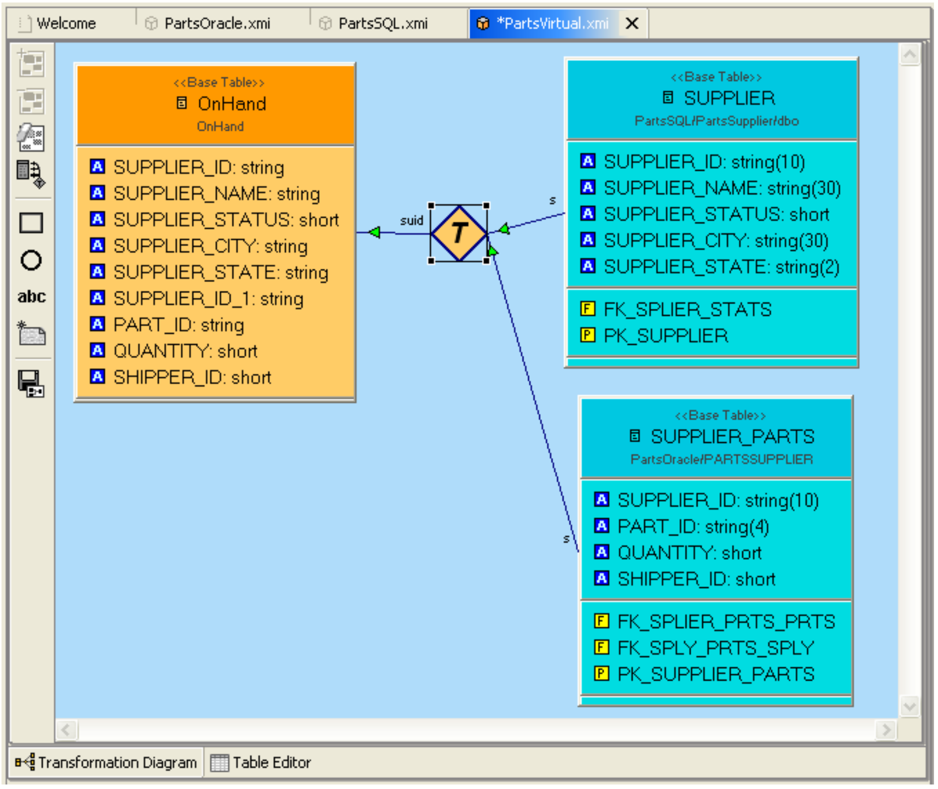
The **Show Categories** button, when toggled on, groups properties into categories.

- The **Problems** and **Message Log** views, located at the bottom right of the Metamodel window, displays errors, warnings, and informational messages about your models. The **Problems** view is updated each time your model projects are built, and contains any violations of metamodel constraints (such as a Relational Table that contains no columns). The **Message Log** contains errors that occur while running the models. Occasionally an error may occur, such as attempting to paste model objects into an illegal container. These errors may result in messages logged to the Message Log view. You can examine the messages for more detailed information.

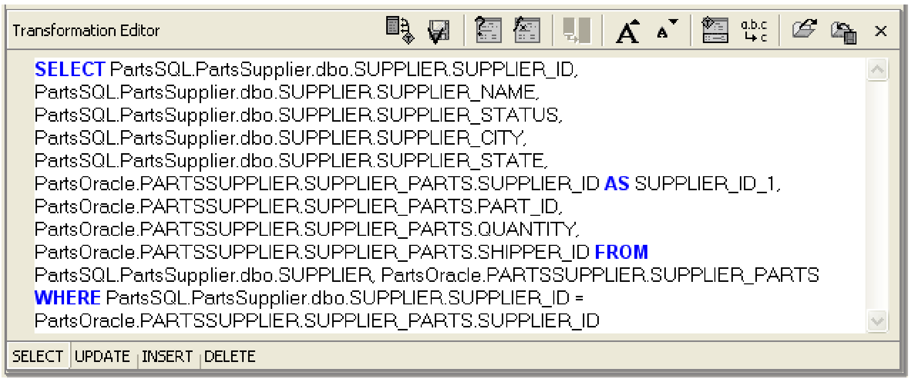


!	Message	Plug-in	Date
✖	Error while Adding '/PartsSupplier/Oracle_Thin_Driver.xmi': File res...	com.metamatrix.modeler.te...	Feb 21, 2004 17:58:53
✖	Error while Adding '/PartsSupplier/TextbookSuppliers.xmi': File reso...	com.metamatrix.modeler.te...	Feb 21, 2004 17:58:52
✖	The Show History doesnt support operating upon {1} resources	com.metamatrix.modeler.te...	Feb 21, 2004 17:41:43
✖	Error while Adding '/PartsSupplier/.project': File resource at the giv...	com.metamatrix.modeler.te...	Feb 21, 2004 16:15:05
✖	Error while Adding '/PartsSupplier/BookDatatypes.xsd': File resourc...	com.metamatrix.modeler.te...	Feb 21, 2004 16:15:05

- The **Editor Panel** view, located on the right, contains a tab at the top for every model file that is open to be viewed or edited.



- The **Transformation Editor** panel appears after double-clicking on the Transformation icon. It defines the structure of the target of your transformation.



```
SELECT PartsSQL.PartsSupplier.dbo.SUPPLIER.SUPPLIER_ID,
PartsSQL.PartsSupplier.dbo.SUPPLIER.SUPPLIER_NAME,
PartsSQL.PartsSupplier.dbo.SUPPLIER.SUPPLIER_STATUS,
PartsSQL.PartsSupplier.dbo.SUPPLIER.SUPPLIER_CITY,
PartsSQL.PartsSupplier.dbo.SUPPLIER.SUPPLIER_STATE,
PartsOracle.PARTSSUPPLIER.SUPPLIER_PARTS.SUPPLIER_ID AS SUPPLIER_ID_1,
PartsOracle.PARTSSUPPLIER.SUPPLIER_PARTS.PART_ID,
PartsOracle.PARTSSUPPLIER.SUPPLIER_PARTS.QUANTITY,
PartsOracle.PARTSSUPPLIER.SUPPLIER_PARTS.SHIPPER_ID FROM
PartsSQL.PartsSupplier.dbo.SUPPLIER, PartsOracle.PARTSSUPPLIER.SUPPLIER_PARTS
WHERE PartsSQL.PartsSupplier.dbo.SUPPLIER.SUPPLIER_ID =
PartsOracle.PARTSSUPPLIER.SUPPLIER_PARTS.SUPPLIER_ID
```

The various view panels do not represent different data. They represent the same metamodel in different ways. You can customize the size of the panels within the MetaBase Modeler to reflect your particular work habits and preferences.

Workspaces and Projects

As you model metadata within the MetaBase Modeler, you work in the MetaBase Modeler window. The window represents a workspace, a distinct environment in which you can open, create, and modify individual models. The **Model Explorer/Outline** panel expands to show the contents of your workspace at any given time.

Before you can populate and manipulate the metadata in a workspace you must create a project. A project represents a named workspace you can open at a later time or add to the MetaBase Repository.

The MetaBase Modeler lets you open more than one workspace (or project) at a time. Each workspace or project displays in a separate MetaBase Modeler window. Note that although you can open multiple workspaces, you can only open one MetaBase Repository Manager window.

MODELING YOUR METADATA

The MetaBase Modeler offers flexibility in modeling your enterprise's information. However, when you first sit down to model metadata, we suggest that you follow these basic steps:

1. **Create a project.** Start by creating a project to encompass all of the metamodels that you will import or create.

2. **Connect your existing EIS systems to the MetaMatrix Modeler.** Create a physical metadata model to describe the information stored in your native enterprise information system (EIS). The physical metadata model contains technical metadata that describes the ultimate sources for any virtual metadata you model. For more information, see "Creating a New Physical Model."

   If your MetaBase Modeler includes a plug-in for the EIS, you can import the metadata information directly from the data source. For more information, see "Importing Metadata."

3. **Import the information source(s) into the MetaMatrix Modeler.** Create virtual metadata models that describe the way your enterprise ultimately uses the data located in your EISes. If your organization uses the MetaMatrix Server for information integration, your applications can query underlying data in your EISes through the virtual metadata; if you're only using MetaBase for metadata management, these virtual metadata models provide useful business metadata representation. For more information, see "Creating a New Virtual Model."

4. **Create the transformation between your existing information source(s) and the new information source you are creating.** Within the virtual metadata models, you can create transformations, which demonstrate how you derive the virtual metadata from the information within your physical EISes. For more information, see "Creating Transformations."

5. **Map the XML tags, if desired.** If you want to describe your information in the terms of an XML document, based on an XML Schema file, you can create a virtual XML document and use transformations and mapping links to map information to the XML. For more information, see "Mapping Other Data Sources to XML."

6. **Validate and test the model.** If you're deploying the metadata model in the MetaMatrix Server, you should validate it to ensure the model contains all necessary metadata. For more information, see "Error Analysis and Rebuilding the Project."

   The validity analysis checks to ensure all required meta object properties have values in order that the model's transformations ultimately link to physical data sources.

   - **Optional: Add your new models and projects to the MetaBase Repository.** You can run your virtual database locally from your own PC, or you can share it by adding it to a MetaBase Repository. The repository provides persistent storage and version control for your metadata models, which enables other members of your organization to review and modify them as well. For more information, see "Sharing Models and Projects in the Team Repository."

The remainder of this guide describes the modeling processes, within the MetaBase Modeler Workspace window in greater detail.

Chapter 5: Creating Metadata Models

When you model your enterprise information systems, you need to capture the essence of how the data resides in your physical data storage. This modeling relies heavily on technical metadata that describes the structures, including the packages, classes, and attributes each data store contains. To contain this structure, you create a physical model. For more information, see "Creating a New Physical Model."

Once you have one or more physical models that describe your underlying data, or even before you have fully modeled your physical data, you can create virtual models to contain the business view, or application view, of the data. These virtual models can contain not only the package, class, and attributes, but also transformations, which determine how your virtual meta objects come from those in a physical model or another virtual model. To contain these meta objects and their transformations, you create a virtual model. For more information, see "Creating a New Virtual Model."

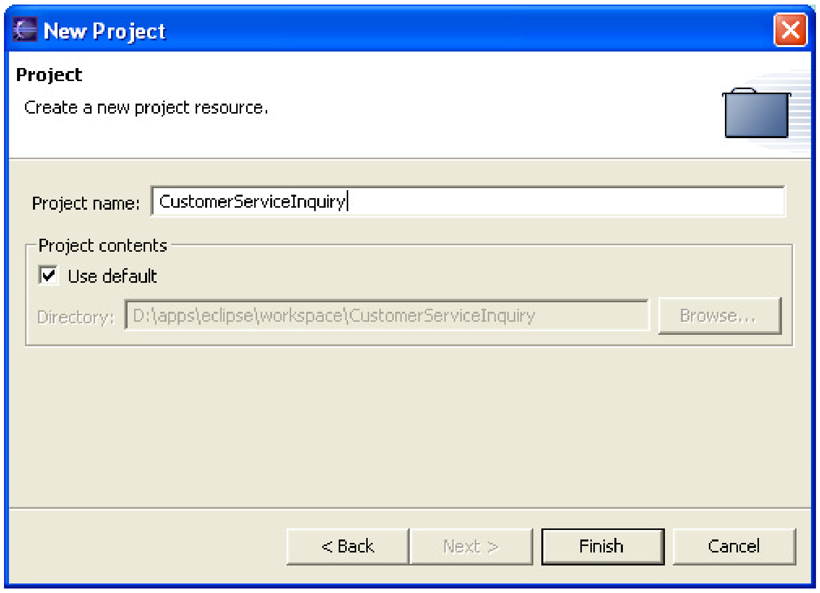
You can also copy an existing model into a new virtual model. For more information, see "Copying an Existing Model."

You can also create new physical metadata models by importing information from a variety of physical data sources. For more information, see the chapter "Importing Metadata."

CREATE A PROJECT

The first step in creating a metamodel is to open a project folder for the model that you will create.

1. To initialize a project, from the menubar select **File >New > Model Project…**
2. The **New Project** dialog box appears.


New Project

Project

Create a new project resource.

Project name: CustomerServiceInquiry

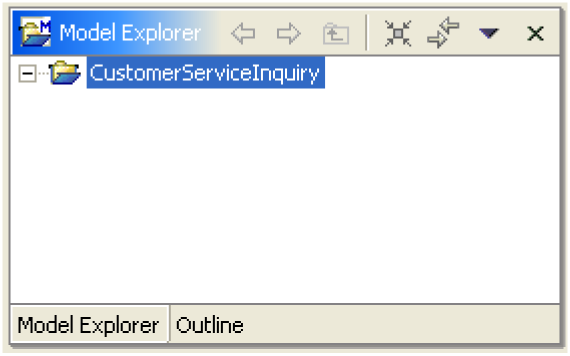
Project contents

☑ Use default

Directory: D:\apps\eclipse\workspace\CustomerServiceInquiry

3. Enter a project name and click the **Finish** button.

4. The project appears in the **Model Explorer/Outline** view.



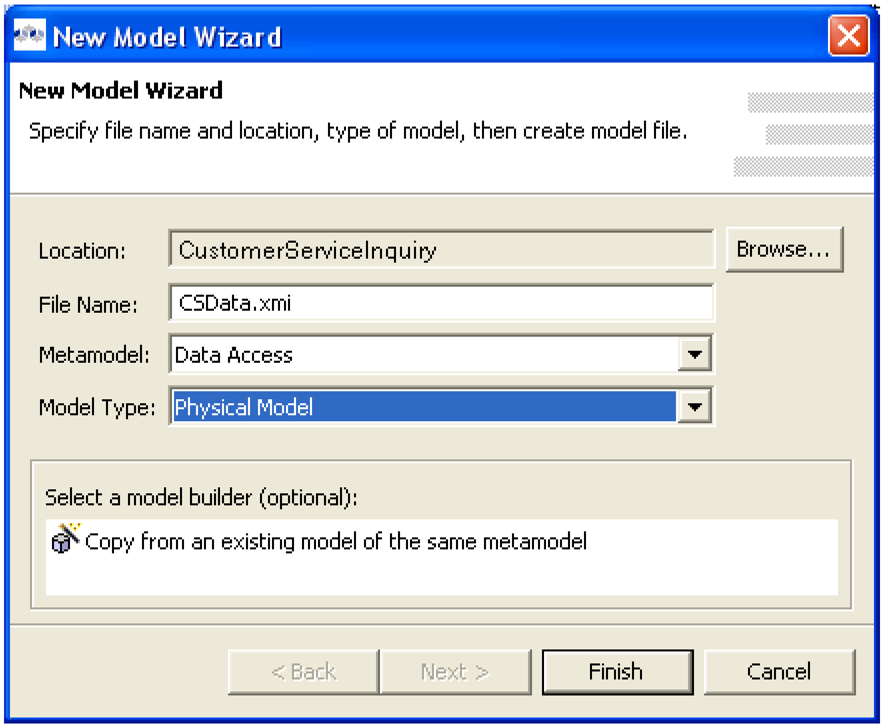
CREATING A NEW PHYSICAL MODEL

You can create a new physical metadata model to describe the physical structure of your enterprise information systems. Physical models, by definition, should match exactly the structure of your native data storage.

The physical metamodel that you create using the Metamatrix Modeler is representative. The MetaMatrix Console manages the actual physical connections to your enterprise information sources**.**

To create a new physical model:

1. From the menubar, select **File > New > Metadata Model…**. The **New Model Wizard** dialog box displays.



New Model Wizard

New Model Wizard

Specify file name and location, type of model, then create model file.

Location: CustomerServiceInquiry

File Name: CSData.xmi

Metamodel: Data Access

Model Type: Physical Model

Select a model builder (optional):

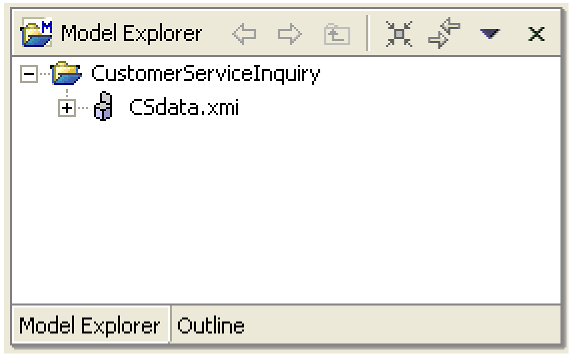
Copy from an existing model of the same metamodel

Input or browse to the **location** of the data source, and select the **file name**.

2. From the **Metamodel** drop-down list, select the metamodel that reflects this enterprise information system.

3. Select **Physical Model** from the **Model Type** dropdown. Click **Finish**.

4. The new model displays on your **Model Explorer/Outline** view bearing a placeholder name such as `newModel1` or **`CSdata.xmi`**. The file also opens in the **Editor** Panel.



NOTE: *You can only save your models to your local directory.*

The MetaBase Modeler creates and saves your new model in the local directory. You can begin modeling in it by creating meta objects. For more information, see the chapter "Creating and Editing Meta Objects."

By creating this model as a physical model, you designate this model as reflecting the physical metadata of your enterprise information system. The contents of the model describe the contents of your enterprise information system.

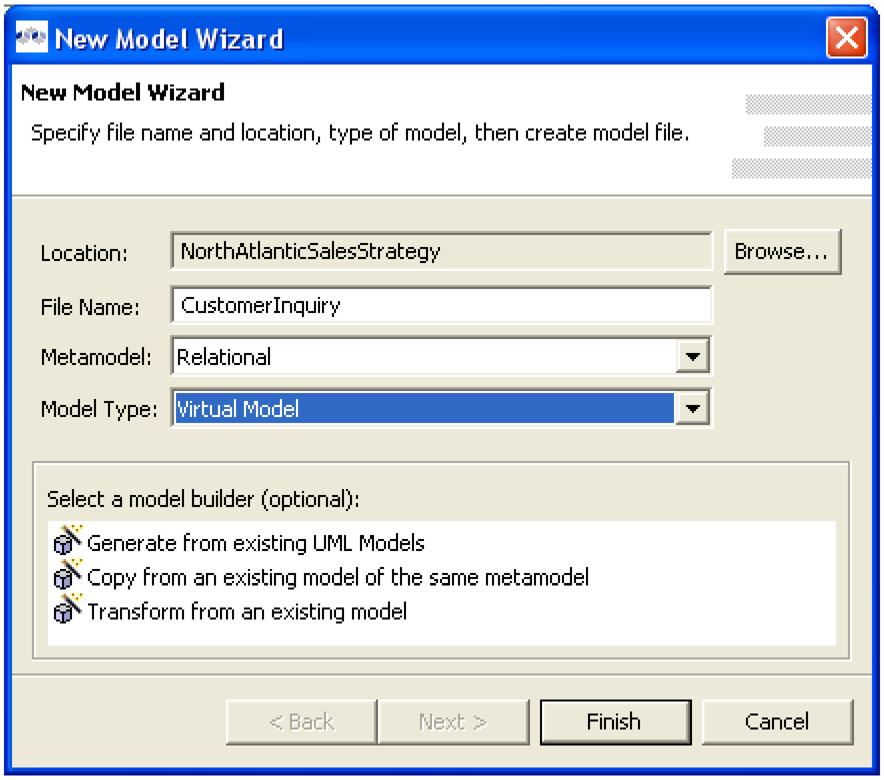
To create your organization's business rules and application view of this enterprise information system, you need to create one or more virtual models that take the information from your physical group and transform it to reflect the business rules your organization applies to its information. For more information about creating these virtual metadata models, see "Creating a New Virtual Model."

CREATING A NEW VIRTUAL MODEL

You can create a new virtual model to describe the business rules and application view of your data. The contents of this model result from transformations that you perform on your physical metadata models or other virtual metadata models.

To create a new virtual model:

1. From the menubar, select **File > New > Metadata Models**.
2. The **New Model Wizard** dialog box displays.



New Model Wizard

Specify file name and location, type of model, then create model file.

Location: NorthAtlanticSalesStrategy

File Name: CustomerInquiry

Metamodel: Relational

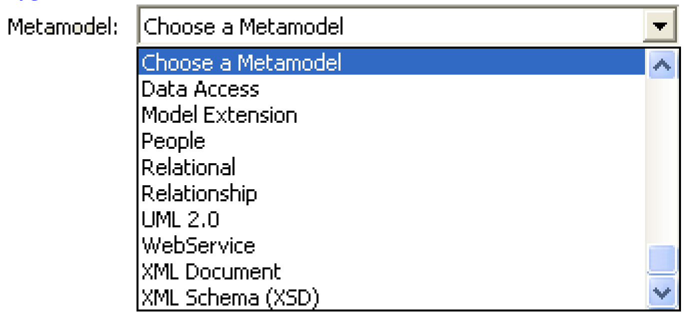
Model Type: Virtual Model

Select a model builder (optional):

Generate from existing UML Models
Copy from an existing model of the same metamodel
Transform from an existing model

Input or browse to the **Location** of the data source, and select the **File Name**.

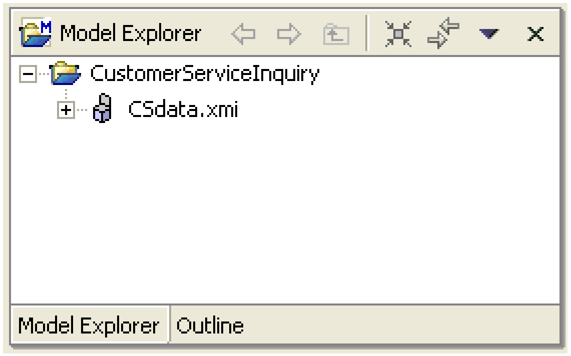
From the **Metamodel** drop-down list, select the metamodel that reflects this enterprise information system. For a complete list of available metamodels, see Available Metamodel Types.



3. Select the **Model Type** drop down for **Virtual Model**.

4. Click **Finish**.

5. The new model displays on your **Model Explorer/Ouline** view bearing a placeholder name such as `newModel1` or `CSdata`. The file also opens in the **Editor Panel** view.



NOTE: You save your models to your local directory under the project location.

The MetaBase Modeler creates and saves your new virtual metadata model. You can begin modeling in it by creating packages, classes, and transformations. For more information, see "Creating and Editing Meta Objects."

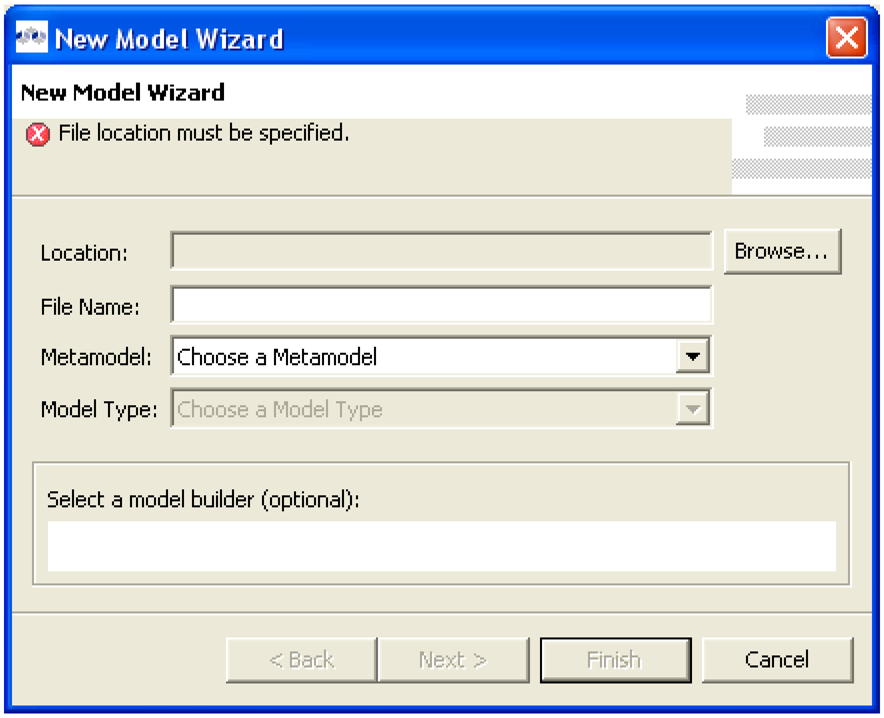
Although the meta objects within the virtual model do not map directly to structures in your enterprise information sources, you must ultimately connect the meta objects, especially attributes, to physical attributes through transformations. If you use the MetaMatrix Server to integrate your data sources, you'll check the validity of these transformations with the Validity Analysis tool.

CREATING A NEW VIRTUAL MODEL FROM AN EXISTING MODEL

You can create a new virtual model by copying an existing model if you have a number of meta objects you want to recycle into your new virtual model. This copy becomes handy if you want to create a more limited or transformed application view of an existing virtual or physical model.

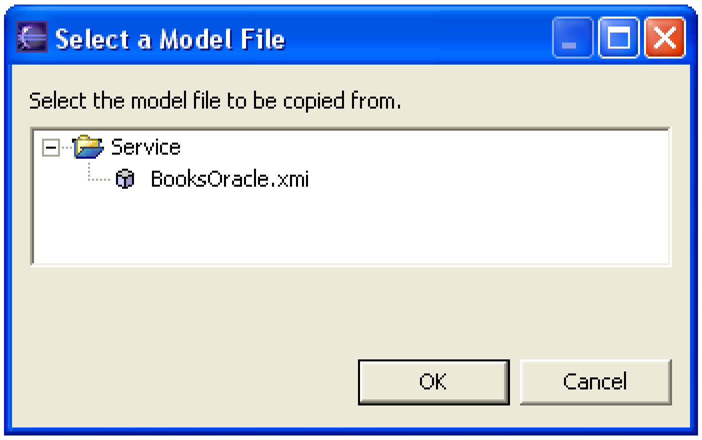
To copy an existing model into a new virtual model:

1. Open the model you want to copy in the workspace.
2. From the menubar, select **File > New > Model…**
3. The **New Model Wizard** dialog box displays.



Input or browse to the **location** of the data source.

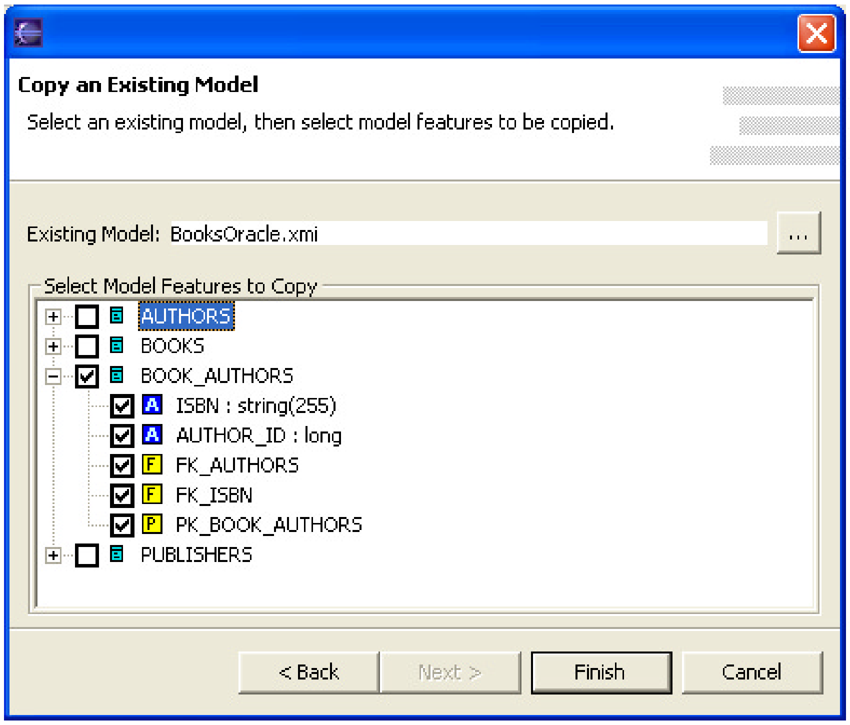
4. From the **Metamodel** drop-down list, select the metamodel that reflects this enterprise information system.
5. Click either the **Virtual Model** or **Physical Model** from the **Model Type** drop down.
6. Select **Copy from an existing model of the same metamodel**.
7. The **Copy an Existing Model** dialog box displays. From your metamodel projects folder (or by browsing to a different folder) you can select the model that you wish to copy into your project.



Select the model file to be copied from.

- Service
  - BooksOracle.xmi

8. The components of the model display. You can select any individual components of the existing model that you wish to copy into your metamodel.


Copy an Existing Model

Select an existing model, then select model features to be copied.

Existing Model: BooksOracle.xmi

Select Model Features to Copy

- [ ] AUTHORS
- [ ] BOOKS
- [✓] BOOK_AUTHORS
  - [✓] ISBN : string(255)
  - [✓] AUTHOR_ID : long
  - [✓] FK_AUTHORS
  - [✓] FK_ISBN
  - [✓] PK_BOOK_AUTHORS
- [ ] PUBLISHERS

The MetaBase Modeler creates and saves your new model in the directory. You can begin modeling in it by creating packages, classes, and transformations. For more information, see "Creating and Editing Meta Objects."

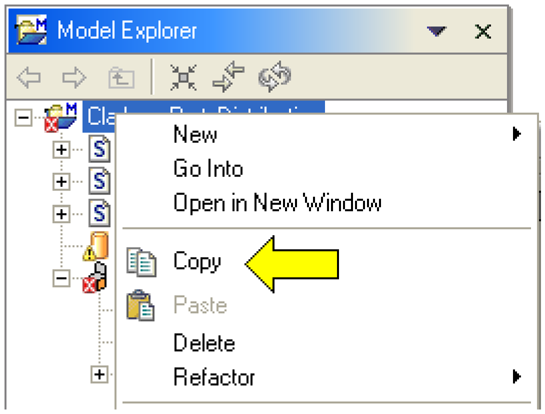
COPYING AN EXISTING WORKSPACE MODEL

You can create a copy of an existing model in the workspace through either of two Modeler actions, **Copy/Paste** and **Save As**.

To execute **Copy** on an existing model in the workspace:

1. Select the model in the explorer tree.

2. Right-click and select **Copy.**



3. Select the modeling project or folder under which the new copy will be located

4. Right-click and select **Paste**. An exact copy of the selected model will be created in the workspace. All references and unique identifiers (UUIDs) within the copy will be the identical to the original.

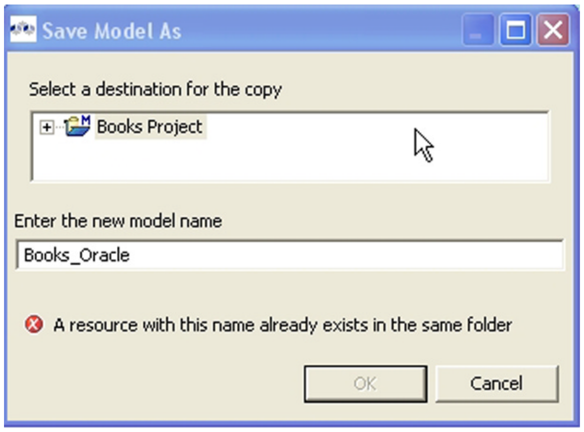
NOTE: Two models with the same unique identifiers are considered ambiguous by the Modeler and using them risks breaking external references between model objects. The copied model should be removed from the workspace or placed within a closed project as soon as possible.

To execute **Save As** on an existing model in the workspace:

5. Open the model in an editor.

6. While the editor is active, from the menubar, select **File > Save As...**

7. Enter the name of the new model in the **Save Model As** dialog and hit OK.



Save Model As

Select a destination for the copy

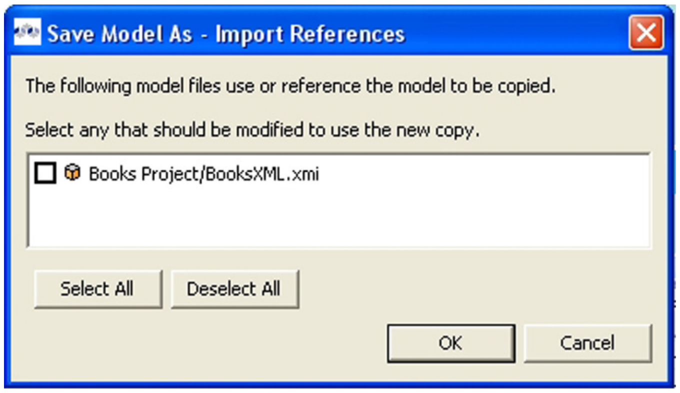
Books Project

Enter the new model name

Books_Oracle

A resource with this name already exists in the same folder

8. If the contents of the model being copied are referenced from other models in the workspace, a **Save Model As - Import References** dialog will appear. The dialog allows the user to specify which, if any, of the of these models should have their references reset to the new copy.



Save Model As - Import References

The following model files use or reference the model to be copied.

Select any that should be modified to use the new copy.

Books Project/BooksXML.xmi

9. Upon completion of the **Save As** action, a new model will be created in the workspace that is an exact copy of the model open in the editor. Unlike the **Copy** action, the model and its contents will be created with new unique identifiers and immediately useable for subsequent modeling activities.

CLOSING A PROJECT

Closing a project makes all models and all objects within those models effectively disappear from the workspace. There is a direct impact on some MetaBase Modeler functionality when closing a project:

- **Build/Validation** is not performed on models in closed projects. When a project is closed, all validation errors and warnings disappear from the Problems view. Also, metadata and relationship **Search** functions will not find objects in closed projects.
- The **Find Model Object** function won't find objects in closed projects.
- Any popup dialog that displays objects, like those in a relationship wizard or the **Set Datatype** dialog, will not display any objects in closed projects.

- Validation on objects in open projects may generate errors if the objects reference other objects that are in closed projects. Specifically, virtual models in open projects that use physical models in closed projects will fail.

Chapter 6: Importing Metadata

THE METADATA IMPORT WIZARD

Purpose of the Wizard

The Metadata Import Wizard helps you create a new model in your workspace by importing metadata information from a physical enterprise information system or other source.

When you import metadata information, the MetaBase Modeler creates a new metadata model for you (in most cases). Once you have created this metadata model, you can alter it as you would any other; however, bear in mind that any changes you make to an imported metadata model do not impact the underlying structure of the enterprise information system the model represents.

You can also use the Metadata Import Wizard, in some cases, to update the information within the models based on changes to the underlying data source. This capability varies based on the type of data source and the plug-in that the Metadata Import Wizard uses.

Import Plug-in Extensibility

The MetaBase Modeler provides an extensible import wizard framework that allows you to create import wizards and plug them into the MetaBase Modeler. For more information about creating your own metadata importers, see the *MetaMatrix MetaBase Plug-in Developer's Guide.*

The MetaBase Modeler comes with several standard plug-ins that you can use to import metadata from common sources. These standard plug-ins let you import data from:

- A JDBC-compliant database. For more information, see "Importing from a JDBC Database."
- The File System loads any file from any location.
- Zip files.

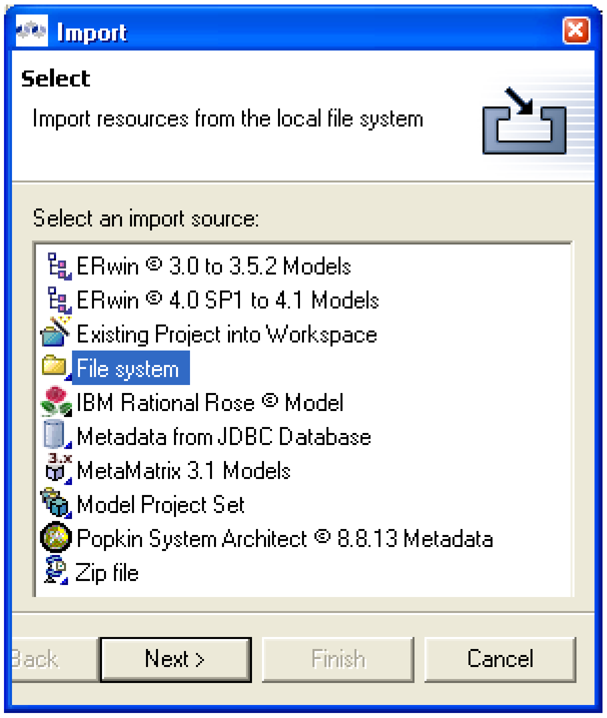
IMPORTING AN XML SCHEMA DOCUMENT

You can import the contents of an XML schema document file, typically stored within a file bearing the .xsd extension, into a metadata model that describes that file. An XML schema document, commonly referred to as an XML schema, acts as a blueprint or list of constraints that describe what elements you can find in an XML instance document that adheres to that XML schema.

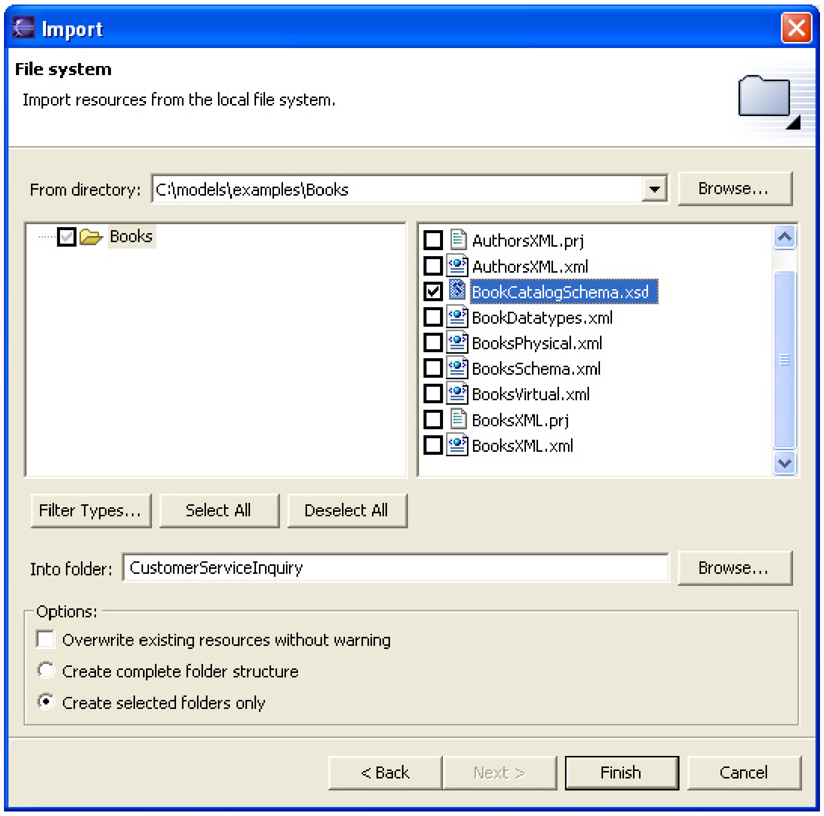
You can import more than one XML schema document into your workspace at a time.

To import one or more XML schema document files:

1. Select **File > Import.** The **Import** wizard displays.
2. Import source from **File System**. Click **Next**.


Import
Select
Import resources from the local file system
Select an import source:
ERwin © 3.0 to 3.5.2 Models
ERwin © 4.0 SP1 to 4.1 Models
Existing Project into Workspace
File system
IBM Rational Rose © Model
Metadata from JDBC Database
MetaMatrix 3.1 Models
Model Project Set
Popkin System Architect © 8.8.13 Metadata
Zip file
Back
Next >
Finish
Cancel

3. Navigate to the folder containing the XML schema document, select the document, and click **Finish**. The new metadata model displays on the **Model Explorer/Outline** view with the name of the XML schema file.



From directory: C:\models\examples\Books

[✓] Books

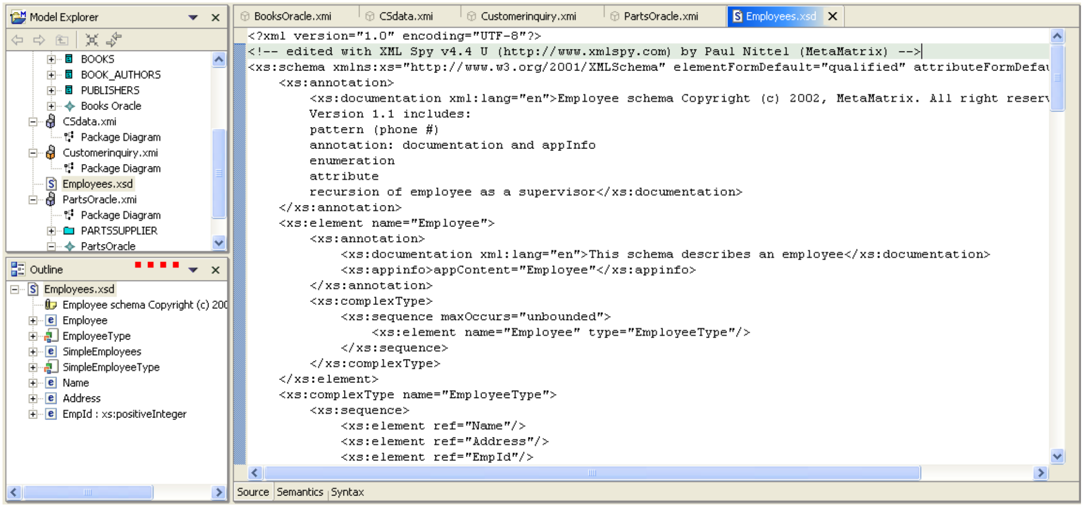
- [ ] AuthorsXML.prj
- [ ] AuthorsXML.xml
- [✓] BookCatalogSchema.xsd
- [ ] BookDatatypes.xml
- [ ] BooksPhysical.xml
- [ ] BooksSchema.xml
- [ ] BooksVirtual.xml
- [ ] BooksXML.prj
- [ ] BooksXML.xml

Into folder: CustomerServiceInquiry

Options:

- [ ] Overwrite existing resources without warning
- ○ Create complete folder structure
- ● Create selected folders only

4. You can make changes to the XML schema document by double-clicking on the .xsd file in the **Model Explorer** view. You can edit the XML schema document in the **Editor Panel** view.



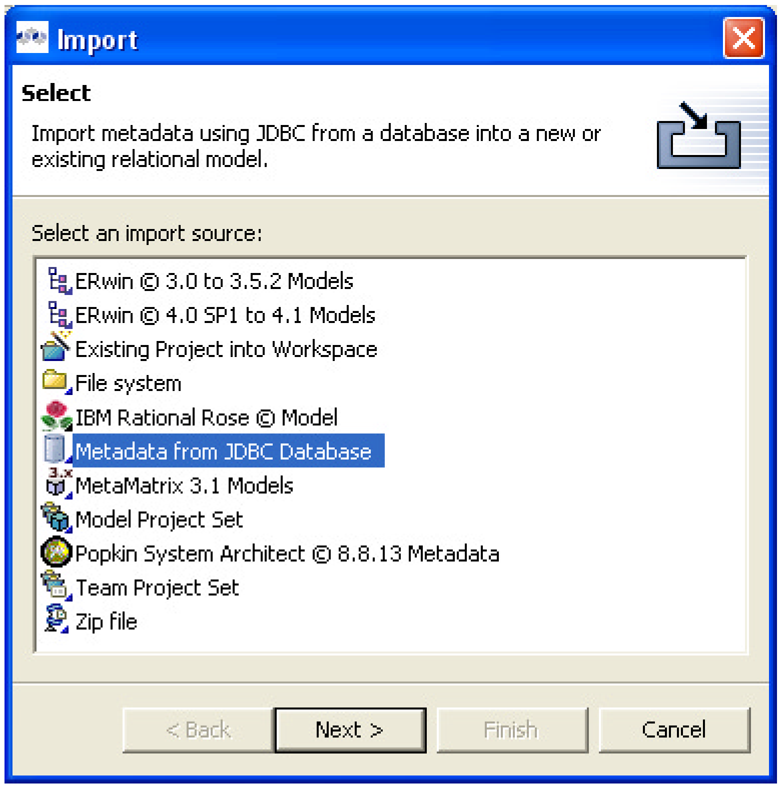
```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.4 U (http://www.xmlspy.com) by Paul Nittel (MetaMatrix) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefau
    <xs:annotation>
        <xs:documentation xml:lang="en">Employee schema Copyright (c) 2002, MetaMatrix. All right reserv
        Version 1.1 includes:
        pattern (phone #)
        annotation: documentation and appInfo
        enumeration
        attribute
        recursion of employee as a supervisor</xs:documentation>
    </xs:annotation>
    <xs:element name="Employee">
        <xs:annotation>
            <xs:documentation xml:lang="en">This schema describes an employee</xs:documentation>
            <xs:appinfo>appContent="Employee"</xs:appinfo>
        </xs:annotation>
        <xs:complexType>
            <xs:sequence maxOccurs="unbounded">
                <xs:element name="Employee" type="EmployeeType"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="EmployeeType">
        <xs:sequence>
            <xs:element ref="Name"/>
            <xs:element ref="Address"/>
            <xs:element ref="EmpId"/>
```

IMPORTING FROM A METAMATRIX MODEL FILE

If you or someone else has exported a metadata model you can easily import that metadata into your workspace or current project. You'll find this useful when you want to import metadata models from other MetaBase Repositories or from a file sent to you by another party or from MetaMatrix Technical Support.

To import an .xmi model file:

1. Select **File > Import.** The Import dialog box displays.
2. Import source from **File System**. Click **Next**.


Import

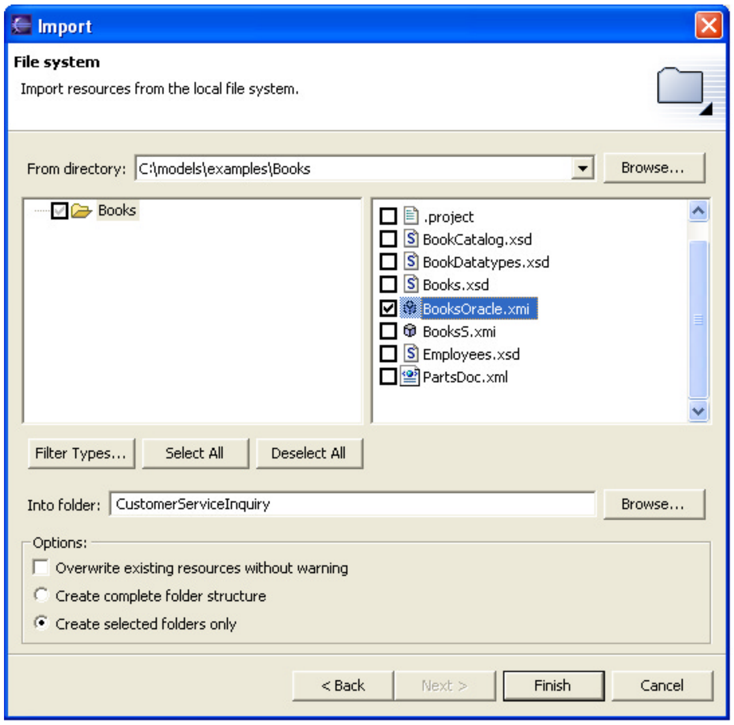
Select

Import metadata using JDBC from a database into a new or existing relational model.

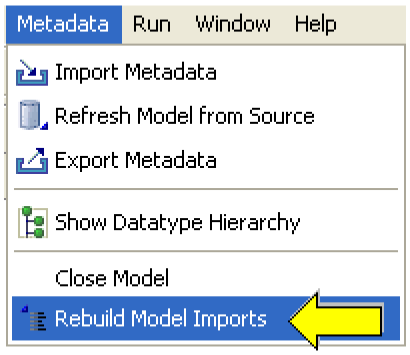
Select an import source:

- ERwin © 3.0 to 3.5.2 Models
- ERwin © 4.0 SP1 to 4.1 Models
- Existing Project into Workspace
- File system
- IBM Rational Rose © Model
- Metadata from JDBC Database
- MetaMatrix 3.1 Models
- Model Project Set
- Popkin System Architect © 8.8.13 Metadata
- Team Project Set
- Zip file

3. Navigate to the folder containing the model file, select the file, and click **Finish**.


Import
File system
Import resources from the local file system.
From directory: C:\models\examples\Books
Browse...
Books
.project
BookCatalog.xsd
BookDatatypes.xsd
Books.xsd
BooksOracle.xmi
BooksS.xmi
Employees.xsd
PartsDoc.xml
Filter Types...
Select All
Deselect All
Into folder: CustomerServiceInquiry
Browse...
Options:
Overwrite existing resources without warning
Create complete folder structure
Create selected folders only
< Back
Next >
Finish
Cancel

4. If you import this model into a different project, you may need to use **Rebuild Model Imports** from the Metadata selection of the menu bar.



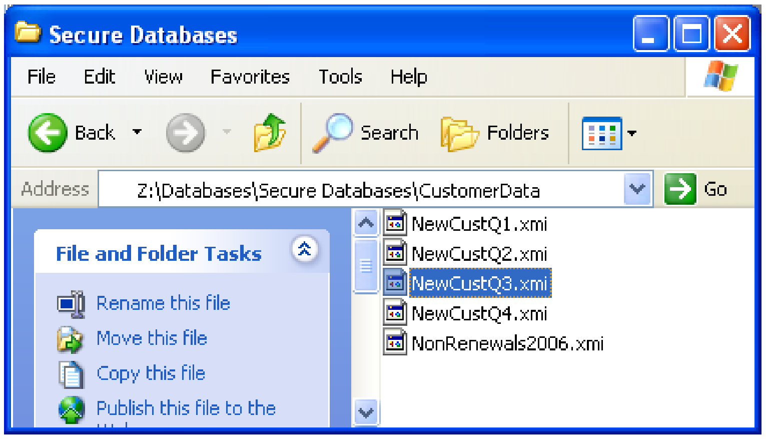
QUICK IMPORT AND COPY OF EXISTING METAMODELS AND XML SCHEMA DOCUMENTS

Copy and Paste Method

You can bypass all of the previous instructions for importing or copying an existing metamodel or XML schema document using a simple copy and paste procedure to get metamodels into an existing project. Most metamodels are files that can be moved from one directory structure to another.

NOTE: You cannot cut and paste JDBC databases. JDBC databases require connections, connection drivers, and passwords.

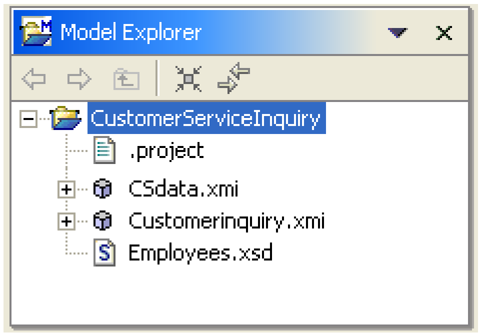
1. Using Internet Explorer, navigate to the model file that you wish to import or copy.



2. Select the model to be imported or copied.

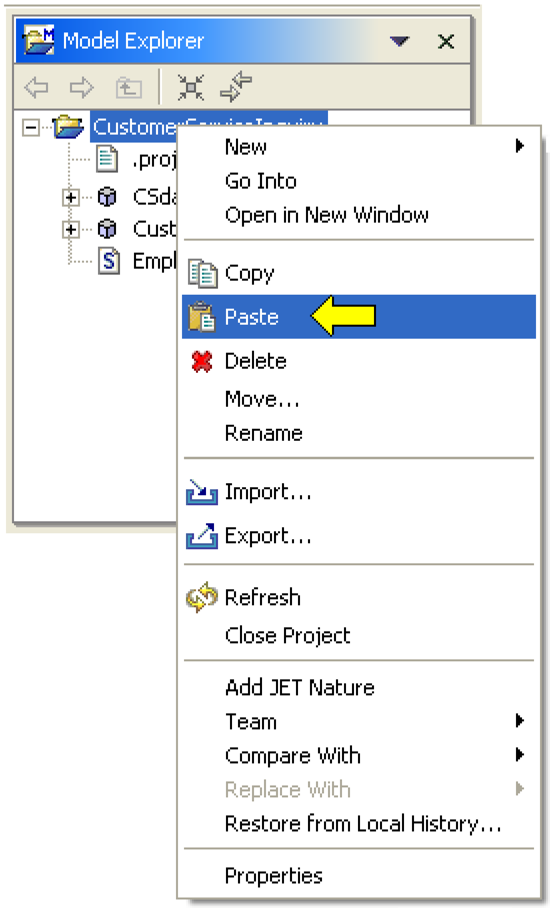
3. Press CTRL+C to copy the model.

4. Open the MetaMatrix Modeler. In the **Model/Explorer** view, right click on the project.

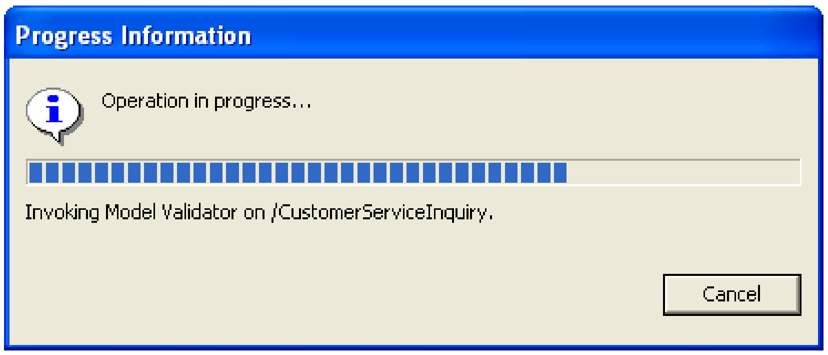


10.

5. Select **Paste** to paste the import or copy into your project. (Paste will be disabled if the Modeler determines that you already have the specified model in your workspace.)



The MetaMatrix Modeler will validate the incoming model and add it to the designated project.



Operation in progress…

Invoking Model Validator on /CustomerServiceInquiry.

IMPORTING FROM A JDBC DATABASE

Using the Metadata Import Wizard

The MetaBase Modeler comes with a number of plug-ins to import metadata from JDBC-compliant databases. You can import metadata information, including schema, tables, views, columns, and keys, directly from these databases. The MetaBase Modeler can read database information and build a metadata model as large or as small as you want.

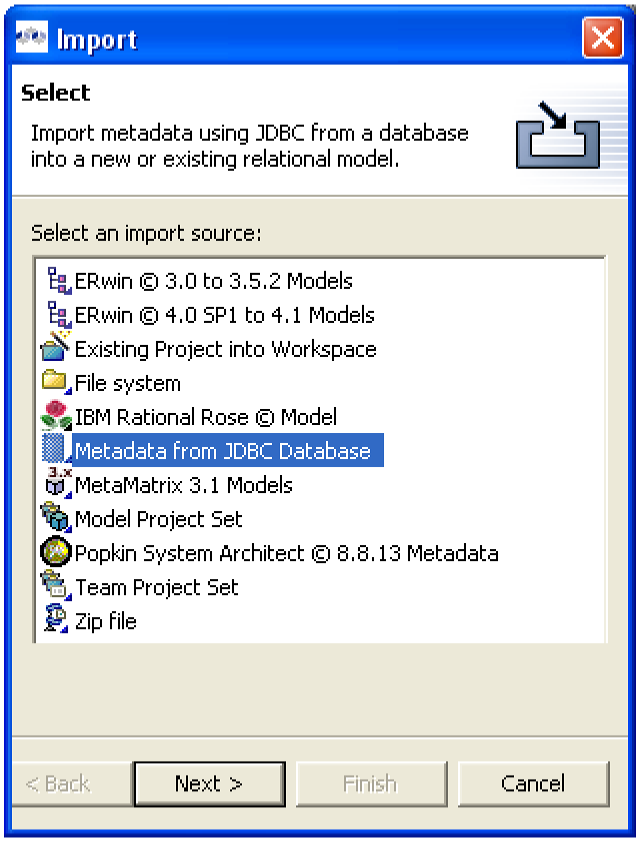
The MetaBase Modeler comes with the following standard JDBC plug-ins:

- General Database.
- IBM DB2 Database.
- IBM DB2 Database with Java Transaction API (JTA) support.
- Informix Database with Java Transaction API (JTA) support.
- Microsoft SQL Server Database
- Microsoft SQL Server Database with JTA support.
- Oracle Database
- Oracle Database with JTA support.
- Sybase Database with JTA support.

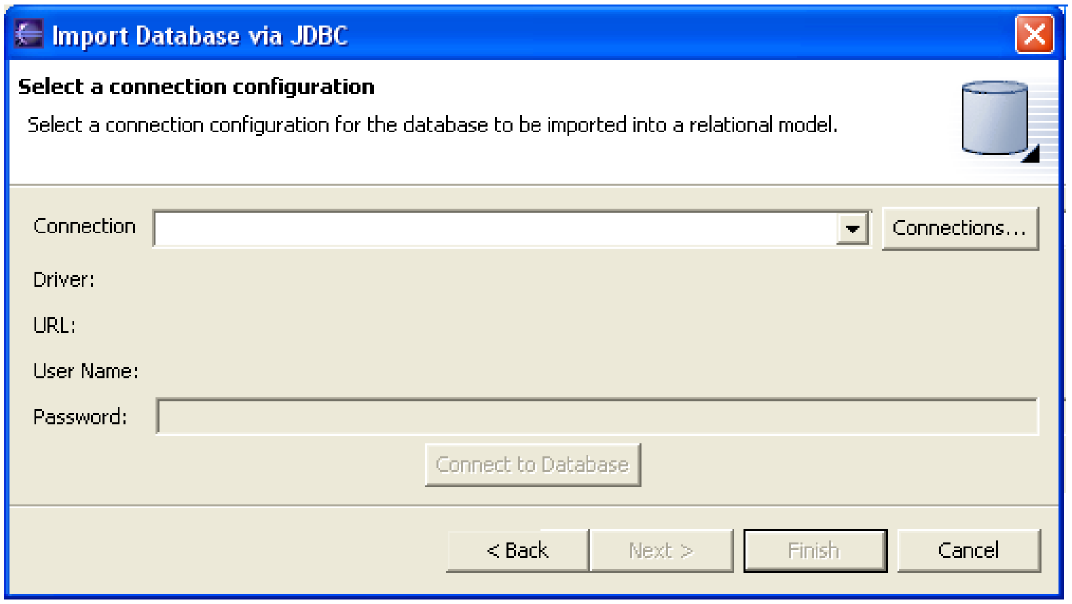
To import metadata from a JDBC database:

1. From the menubar, select **File > Import.** The **Import** dialog box displays.

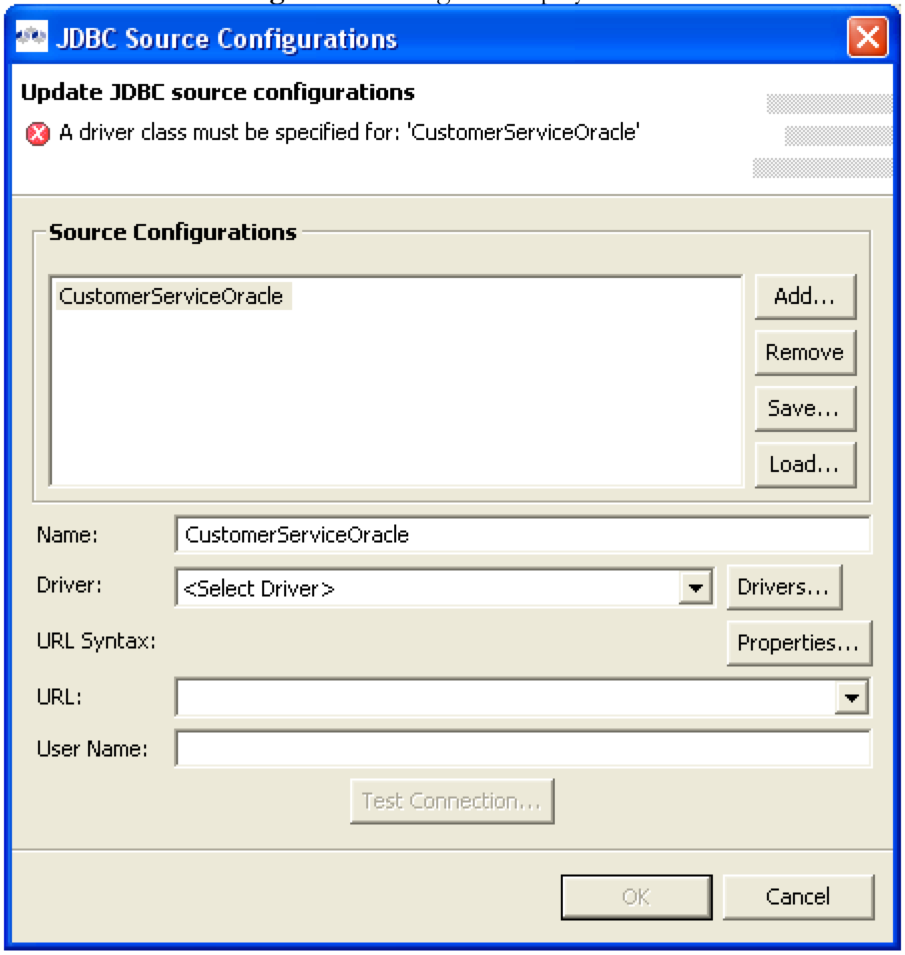
2. Import source from **Metadata from JDBC Database**. Click **Next**.


Import
Select
Import metadata using JDBC from a database into a new or existing relational model.
Select an import source:
ERwin © 3.0 to 3.5.2 Models
ERwin © 4.0 SP1 to 4.1 Models
Existing Project into Workspace
File system
IBM Rational Rose © Model
Metadata from JDBC Database
MetaMatrix 3.1 Models
Model Project Set
Popkin System Architect © 8.8.13 Metadata
Team Project Set
Zip file
< Back
Next >
Finish
Cancel

3. The **Import Database via JDBC** dialog box displays. Click the **Connections** button.


Import Database via JDBC
Select a connection configuration
Select a connection configuration for the database to be imported into a relational model.
Connection
Connections...
Driver:
URL:
User Name:
Password:
Connect to Database
< Back
Next >
Finish
Cancel

4. The **Connection Configuration** dialog box displays.



JDBC Source Configurations

Update JDBC source configurations

A driver class must be specified for: 'CustomerServiceOracle'

Source Configurations

CustomerServiceOracle

Name: CustomerServiceOracle

Driver: <Select Driver>

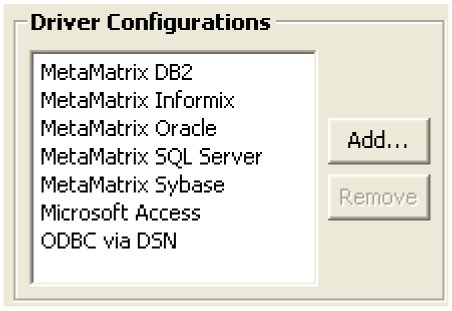
URL Syntax:

URL:

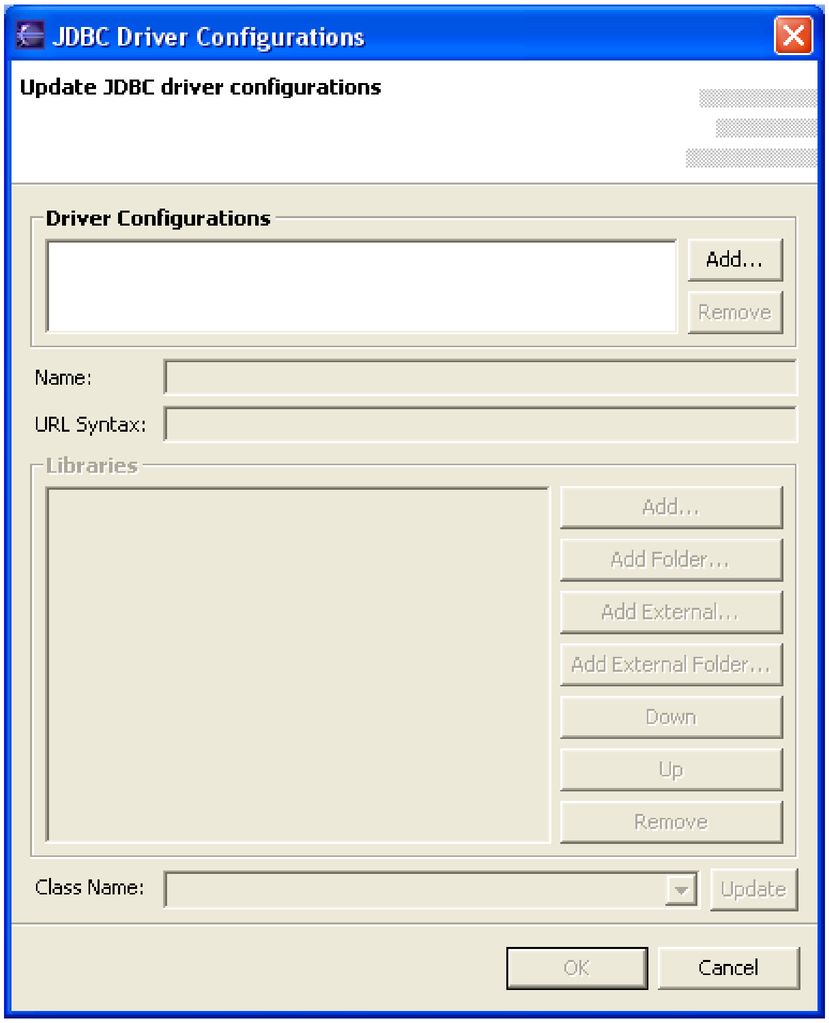
User Name:

5. Enter a name for the database connection that you are initiating. This will be the default name for the model you will import. It is recommended that you make this connection name descriptive. Click the **Add…** button.

6. You may get a list of available **Drivers:** in a dropdown list. If you can select the driver you need from a pre-loaded list, select the driver and **skip to Step 13**. Steps 7 through 12 explain how to initiate a driver for the first time.



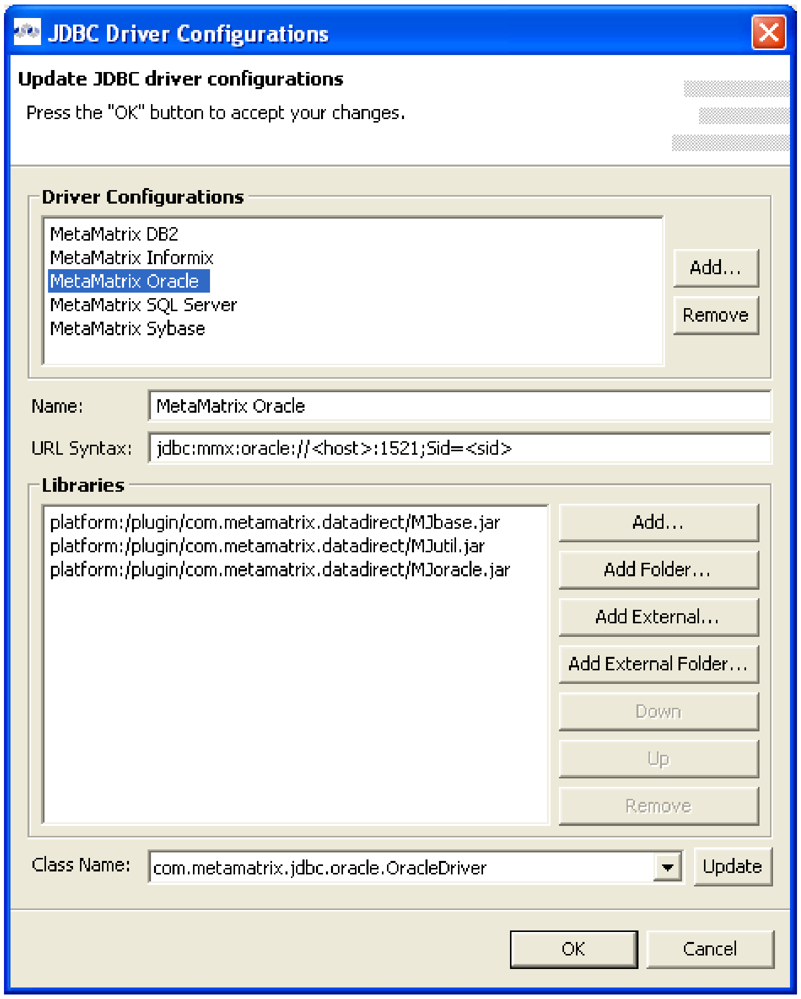
7. If you need to initiate a driver from scratch, follow the instructions in steps 7 through 13. Otherwise skip to step 14. Click the **Drivers…** button. The JDBC Drivers Configuration dialog box displays.



8. Click the **Add…** button. You can now enter the name of the driver configuration in the **Name:** field.

9. In the **URL Syntax:** field, enter the syntax format for your database server connection properties. You will need to load the class path (and related .jar files) for your JDBC drivers. See the documentation that came with your JDBC for specifics on which files are required.

   To load additional plug-ins or files, use the **Add…**, **Add Folder…**, **Add External…**, or **Add External Folder…** buttons at the bottom right of the dialog box to browse to the necessary components specified in the JDBC driver documentation.



MetaMatrix DB2
MetaMatrix Informix
MetaMatrix Oracle
MetaMatrix SQL Server
MetaMatrix Sybase

Name: MetaMatrix Oracle
URL Syntax: jdbc:mmx:oracle://<host>:1521;Sid=<sid>

platform:/plugin/com.metamatrix.datadirect/MJbase.jar
platform:/plugin/com.metamatrix.datadirect/MJutil.jar
platform:/plugin/com.metamatrix.datadirect/MJoracle.jar

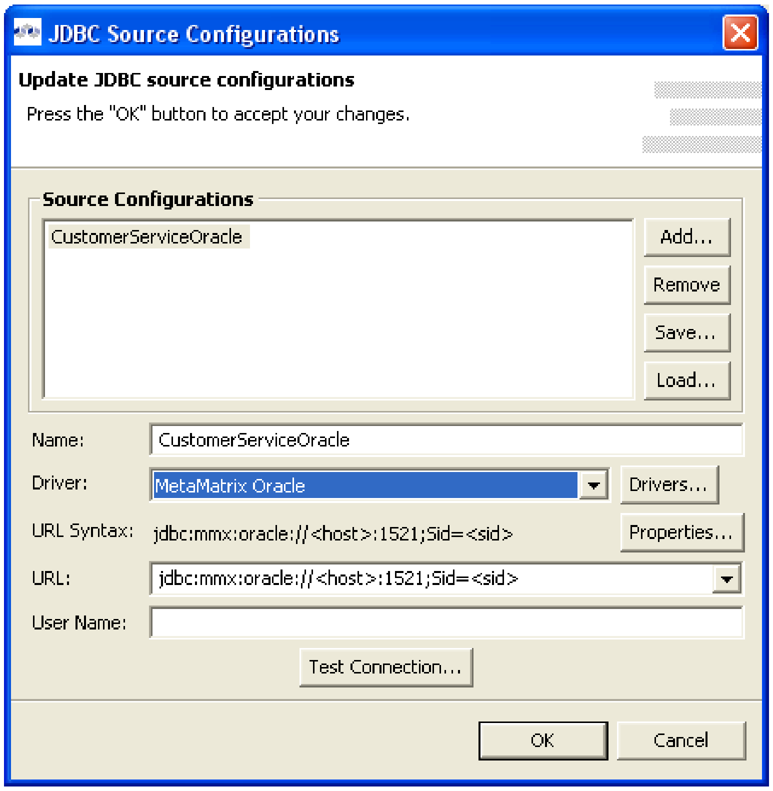
Class Name: com.metamatrix.jdbc.oracle.OracleDriver

10. Click the **Update** button at the bottom right of the dialog box.

11. Select a class name from the **Class Name:** dropdown box.

12. Click **OK**.

13. The **Connection Configurations** dialog box displays again, this time with most of the critical fields completed.


JDBC Source Configurations

Update JDBC source configurations

Press the "OK" button to accept your changes.

Source Configurations

CustomerServiceOracle

Name: CustomerServiceOracle

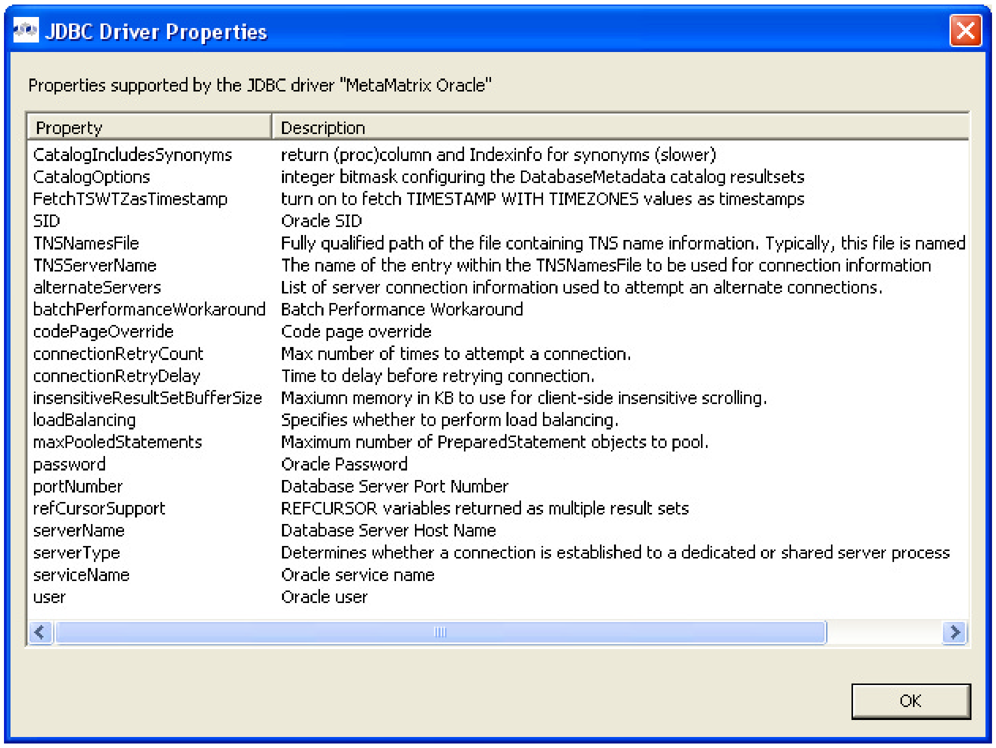
Driver: MetaMatrix Oracle

URL Syntax: jdbc:mmx:oracle://<host>:1521;Sid=<sid>

URL: jdbc:mmx:oracle://<host>:1521;Sid=<sid>

User Name:

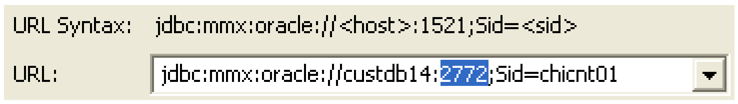
14. The **Properties** button opens a list of the properties associated with the type of JDBC driver that you are using.


JDBC Driver Properties

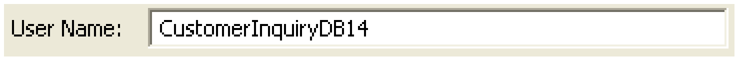
Properties supported by the JDBC driver "MetaMatrix Oracle"

Property	Description
CatalogIncludesSynonyms	return (proc)column and Indexinfo for synonyms (slower)
CatalogOptions	integer bitmask configuring the DatabaseMetadata catalog resultsets
FetchTSWTZasTimestamp	turn on to fetch TIMESTAMP WITH TIMEZONES values as timestamps
SID	Oracle SID
TNSNamesFile	Fully qualified path of the file containing TNS name information. Typically, this file is named
TNSServerName	The name of the entry within the TNSNamesFile to be used for connection information
alternateServers	List of server connection information used to attempt an alternate connections.
batchPerformanceWorkaround	Batch Performance Workaround
codePageOverride	Code page override
connectionRetryCount	Max number of times to attempt a connection.
connectionRetryDelay	Time to delay before retrying connection.
insensitiveResultSetBufferSize	Maxiumn memory in KB to use for client-side insensitive scrolling.
loadBalancing	Specifies whether to perform load balancing.
maxPooledStatements	Maximum number of PreparedStatement objects to pool.
password	Oracle Password
portNumber	Database Server Port Number
refCursorSupport	REFCURSOR variables returned as multiple result sets
serverName	Database Server Host Name
serverType	Determines whether a connection is established to a dedicated or shared server process
serviceName	Oracle service name
user	Oracle user

15. In the **URL:** field, use the prefilled syntax structure to enter the location of the database you are importing. Replace any generic fields with specific server locations, port numbers, or system identifiers.

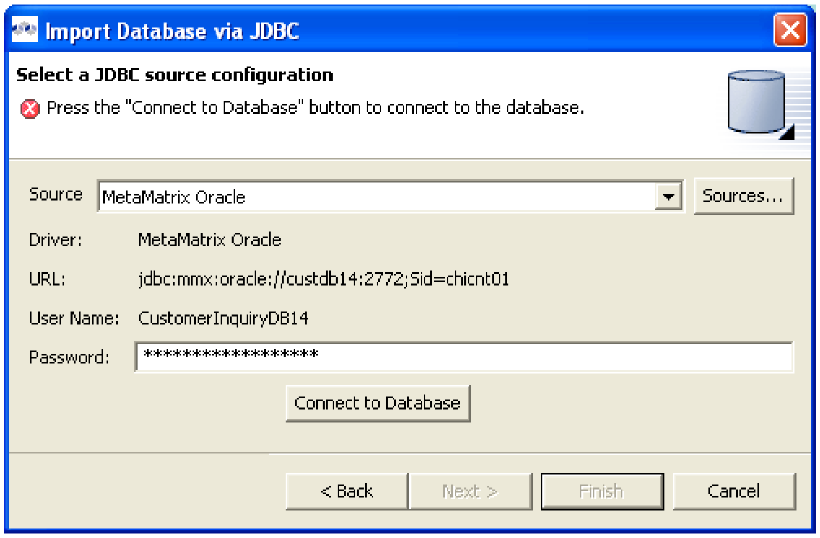

URL Syntax: jdbc:mmx:oracle://<host>:1521;Sid=<sid>
URL: jdbc:mmx:oracle://custdb14:2772;Sid=chicnt01

16. In the **User Name:** field, input the ***name of the database connection to be used to import the database.***


User Name: CustomerInquiryDB14

17. Click **Test Connection**. You may be prompted for a password.

18. Once again, you are returned to the **Import Database via JDBC** dialog box, now with some fields prefilled.



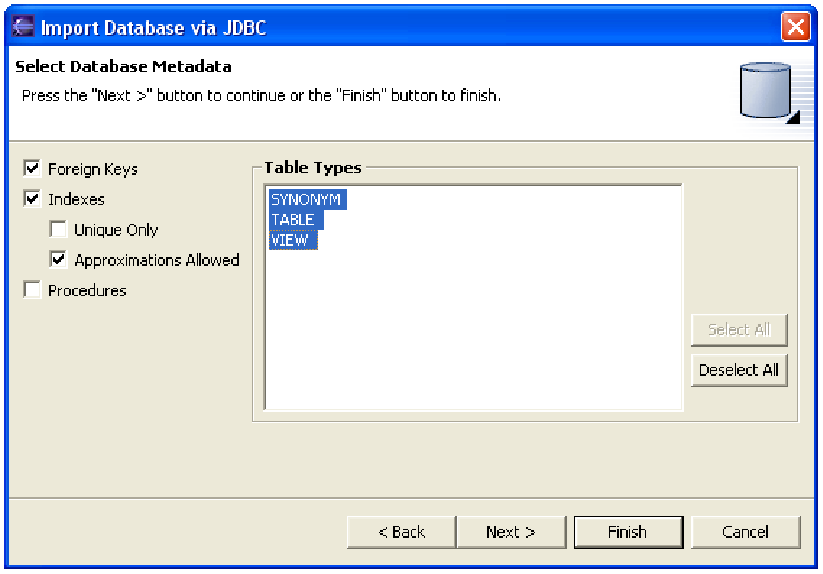
Select a JDBC source configuration
Press the "Connect to Database" button to connect to the database.

Source: MetaMatrix Oracle
Driver: MetaMatrix Oracle
URL: jdbc:mmx:oracle://custdb14:2772;Sid=chicnt01
User Name: CustomerInquiryDB14
Password: ******************

Enter the database connection **Password** and click **Connect to Database**.

19. Once the connection is established, click **Next**.

20. For an Oracle database, this screen might look like this:


Import Database via JDBC

Select Database Metadata

Press the "Next >" button to continue or the "Finish" button to finish.

- [✓] Foreign Keys
- [✓] Indexes
  - [ ] Unique Only
  - [✓] Approximations Allowed
- [ ] Procedures

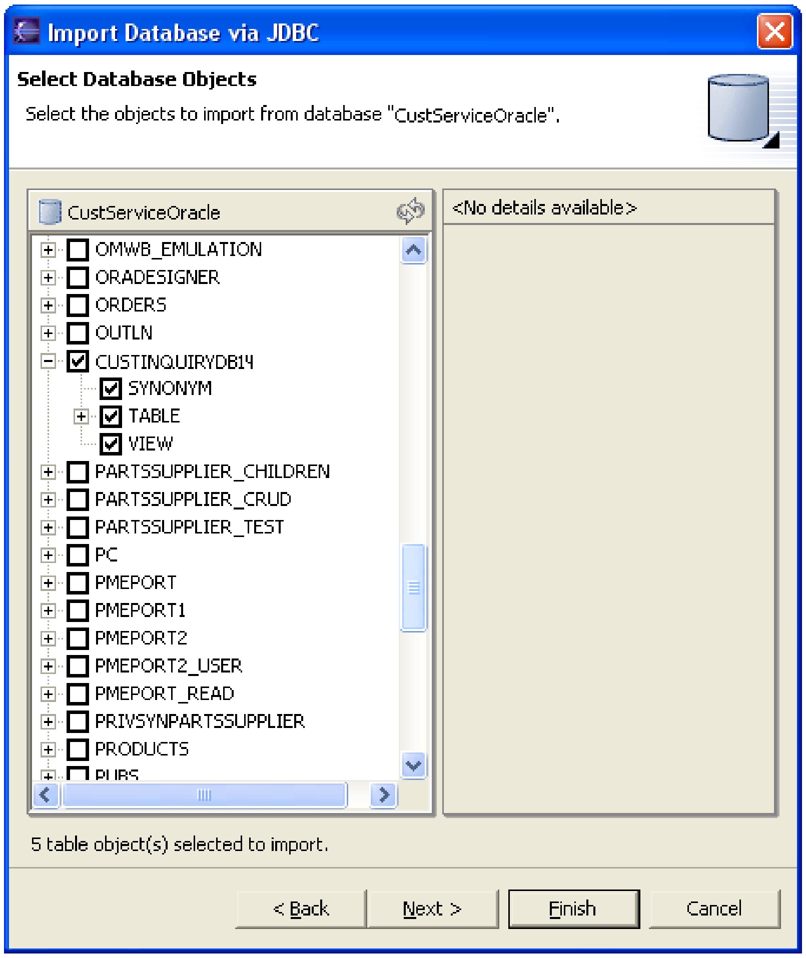
Table Types: SYNONYM, TABLE, VIEW (selected)

You can choose the meta objects you want to import. From the **Tables Types** list box, select the types of features in the database you want to import. The types of tables available differ depending upon the type of database.

21. You can click the following checkboxes:

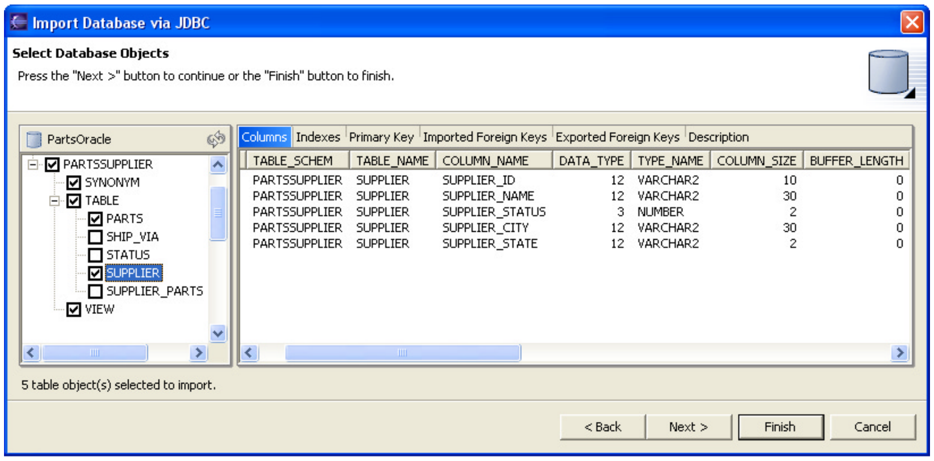
- **Foreign Keys** to import foreign keys.
- **Indexes** to import indexes.
- **Unique Only** to import only unique indexes.
- **Approximations Allowed** to allow approximations of index information, such as number of rows, when importing indexes.
- **Procedures** to import stored procedures.

22. Click **Next**. The following dialog box appears.



This screen shows metadata in the context of your connection and previous selections. You can see metadata for other connection accounts, but no data.

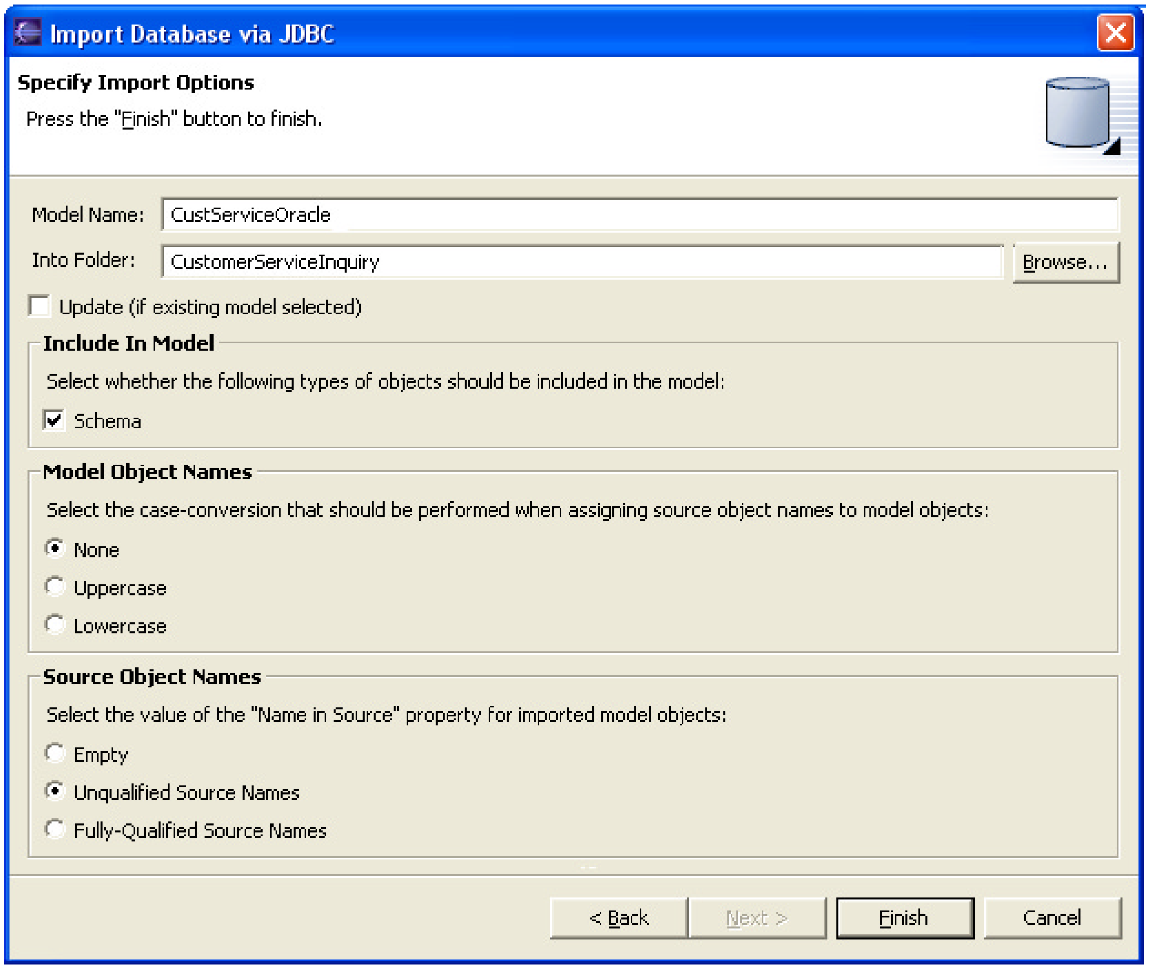
23. To see additional metadata for the connection that you are establishing, click on the checked description in directory tree. You can see additional details such as column headings, indexes, keys, and descriptions.



TABLE_SCHEM	TABLE_NAME	COLUMN_NAME	DATA_TYPE	TYPE_NAME	COLUMN_SIZE	BUFFER_LENGTH
PARTSSUPPLIER	SUPPLIER	SUPPLIER_ID	12	VARCHAR2	10	0
PARTSSUPPLIER	SUPPLIER	SUPPLIER_NAME	12	VARCHAR2	30	0
PARTSSUPPLIER	SUPPLIER	SUPPLIER_STATUS	3	NUMBER	2	0
PARTSSUPPLIER	SUPPLIER	SUPPLIER_CITY	12	VARCHAR2	30	0
PARTSSUPPLIER	SUPPLIER	SUPPLIER_STATE	12	VARCHAR2	2	0

For more information on the significance of data type, type name, and their relationship to JDBC type in the established JDBC import, see *Appendix C*, "JDBC Imports and Built-in Datatypes."

24. Click **Next**. The next JDBC Import dialog box displays.



Import Database via JDBC

Specify Import Options

Press the "Finish" button to finish.

Model Name: CustServiceOracle

Into Folder: CustomerServiceInquiry

☐ Update (if existing model selected)

Include In Model

Select whether the following types of objects should be included in the model:

☑ Schema

Model Object Names

Select the case-conversion that should be performed when assigning source object names to model objects:

[✓ None] Uppercase Lowercase

Source Object Names

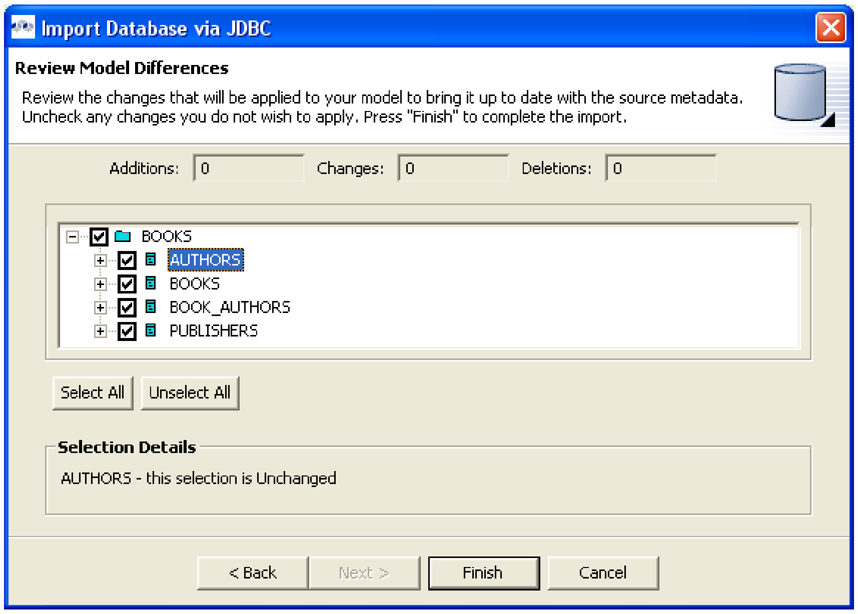
Select the value of the "Name in Source" property for imported model objects:

Empty [✓ Unqualified Source Names] Fully-Qualified Source Names

Where the previous JDBC import dialog boxes dealt with *which* metadata will be imported, this panel specifies *how* metadata will be imported. Following is an overview of the options available through this dialog box.

- **Update** - Select update if you want this import to update a model that already exists in your workspace. **If you check the Update box,** there will be one additional screen before the importer wizard completes. You will only see this screen if you have previously imported the same model

☐ Update (if existing model selected)


Import Database via JDBC

Review Model Differences

Review the changes that will be applied to your model to bring it up to date with the source metadata. Uncheck any changes you do not wish to apply. Press "Finish" to complete the import.

Additions: 0 Changes: 0 Deletions: 0

- [✓] BOOKS
  - [✓] AUTHORS
  - [✓] BOOKS
  - [✓] BOOK_AUTHORS
  - [✓] PUBLISHERS

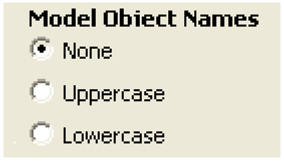
Selection Details

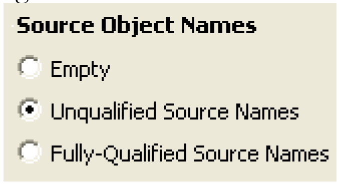
AUTHORS - this selection is Unchanged

From this screen you can see at-a-glance the differences between the existing version of the model, and the reimported version. You can deselect any attributes that you do not wish to import.

- **Schema** - Keep the original format or nested folders of metadata upon import.

[✓] Schema

- **Model Object Names** – Select any conversion of case-shifting in importing metadata naming conventions.

Model Object Names
[✓ None] Uppercase Lowercase

- **Source Object Names** – Standardizes non-standardized naming, and reconciles "name" against "name in source."

Source Object Names
Empty [✓ Unqualified Source Names] Fully-Qualified Source Names

25. Click **Finish**. The new metadata model displays on the **Model Explorer/Outline** tab with the database you imported.

IMPORTER FOR ERWIN 3.5.2 MODELS

MetaMatrix provides an importer for ERwin files (version 3.5.2). The importer will create Relational models in the MetaMatrix workspace.

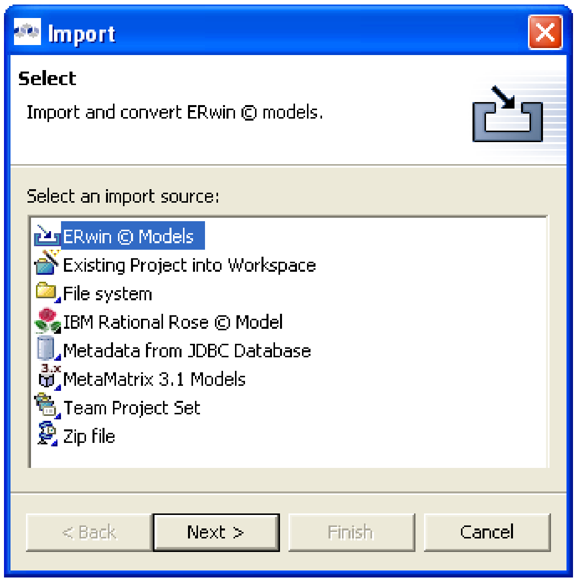
Supported structural features of ERwin models will be:

- Schema
- Catalog
- Table
- View
- Column
- Index
- Procedure
- Parameter
- Foreign key
- Primary key
- Unique key
- Logical relationship

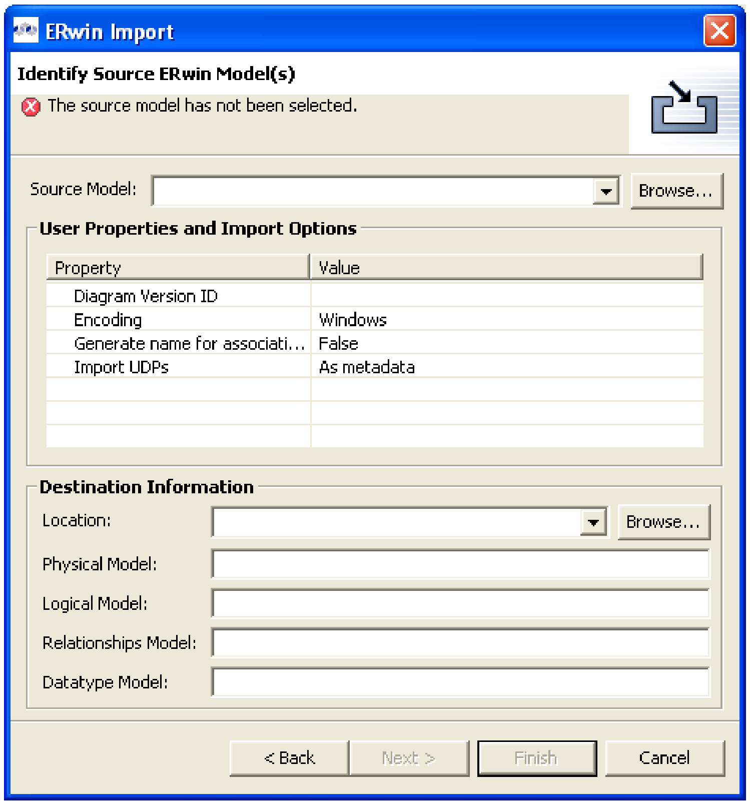
Any properties (name-value pairs) of these features that are unique your use of ERwin (or describe physical database attributes) will be implemented using the Modeler's metadata extension functionality. The Modeler will allow the display and editing of the imported metadata in UML notation in package diagrams within the model.

To import and convert an ERwin model, use the following procedure.

1. From the MetaBase Modeler menubar, select **File > Import**. The **Import** selection screen displays.



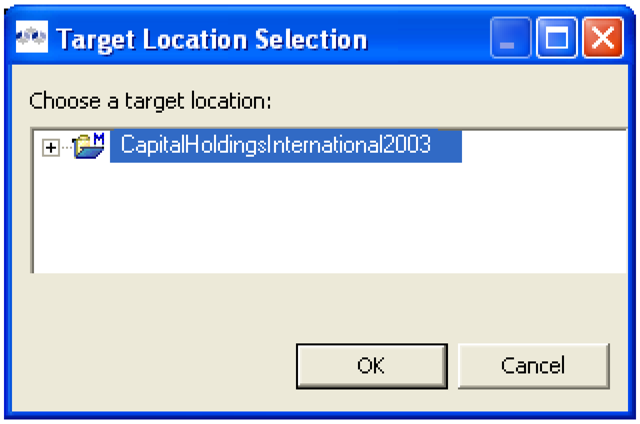
2. Select **ERwin Models** as your import source. Click **Next**. The **ERwin Import** dialog box displays.



The source model has not been selected.

Property	Value
Diagram Version ID	
Encoding	Windows
Generate name for associati...	False
Import UDPs	As metadata

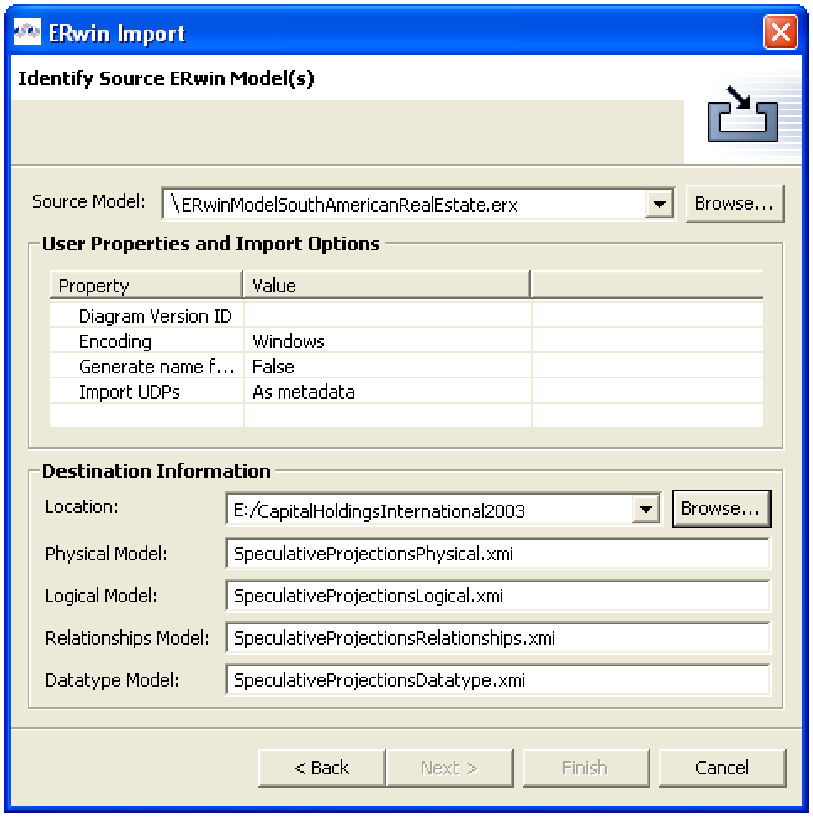
3. Select an ERwin source model. Use the **Browse** button to navigate, if necessary.



Choose a target location:

CapitalHoldingsInternational2003

4. Complete the dialog boxes on the ERwin Import Screen. Any user-defined properties and values will be at the top of the information listed in the **User Properties and Import Options** section. A number of core properties and values will always appear.



Source Model: \ERwinModelSouthAmericanRealEstate.erx

User Properties and Import Options

Property	Value
Diagram Version ID	
Encoding	Windows
Generate name f...	False
Import UDPs	As metadata

Destination Information

Location:	E:/CapitalHoldingsInternational2003
Physical Model:	SpeculativeProjectionsPhysical.xmi
Logical Model:	SpeculativeProjectionsLogical.xmi
Relationships Model:	SpeculativeProjectionsRelationships.xmi
Datatype Model:	SpeculativeProjectionsDatatype.xmi

5. Click **Finish**.

When you import an ERwin model, all entities from the ERwin model that are **Logical Only** or **Logical and Physical** get created in the Logical (UML) model.

Also, all entities from the ERwin model that are **Physical Only** or **Physical and Logical** get created in the Physical model.

All types used are created in a **Datatype** schema. All relationships between entities (Primary Keys, Foreign Keys, Associations, etc) are represented in the Relationship model and may also be represented in another model.

IMPORTER FOR RATIONAL ROSE MODELS

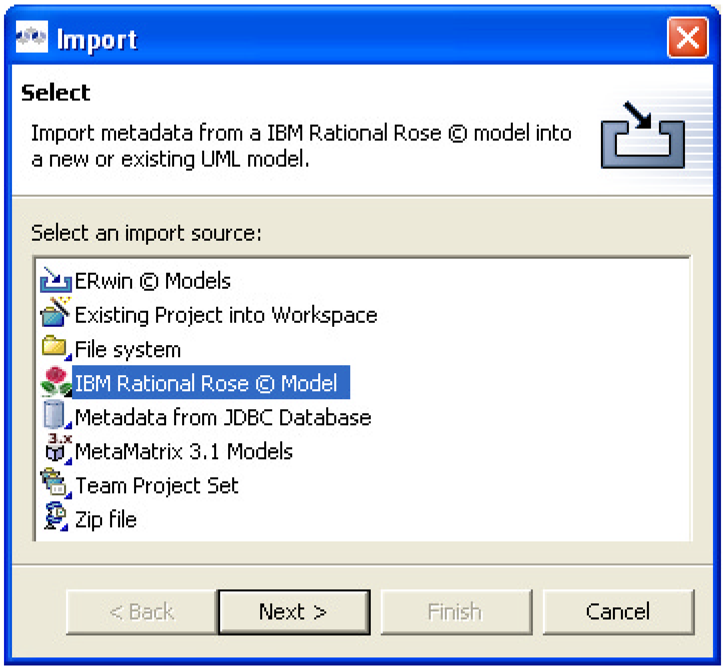
MetaMatrix provides an importer for files exported from Rational Rose versions 2000 and 2001.

The importer creates UML models in the MetaMatrix workspace. Supported structural features of UML will be defined by the UMLClasses Metamodel Project. The content of the Rose model file that matches these supported features of the MetaMatrix UML metamodel are imported.

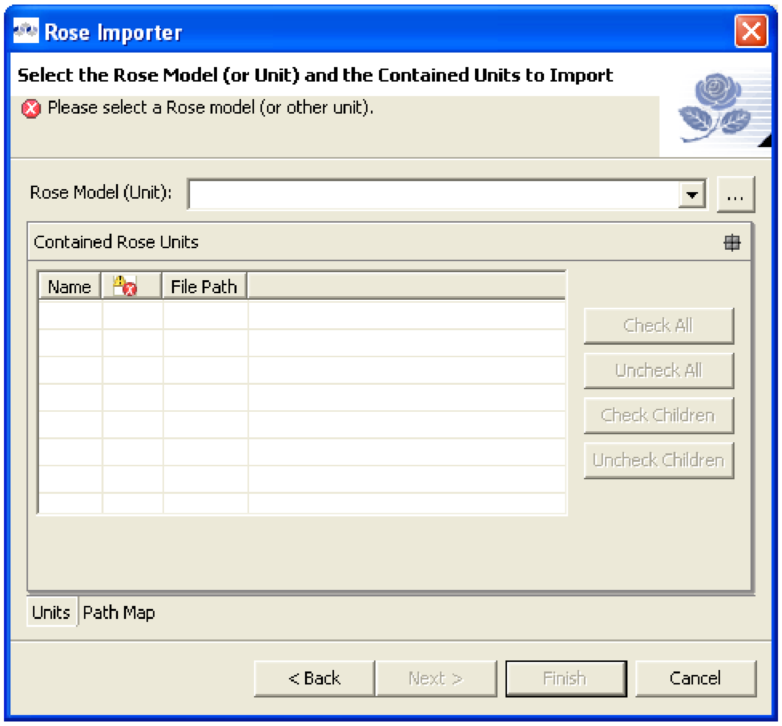
The importer allows subsequent reimportation of the Rose model file. It incorporate changes into the MetaMatrix UML model. Any properties of the supported UML constructs that are unique to your use of Rose are implemented using the Modeler's metadata extension functionality.

To import and convert a Rational Rose model, use the following procedure.

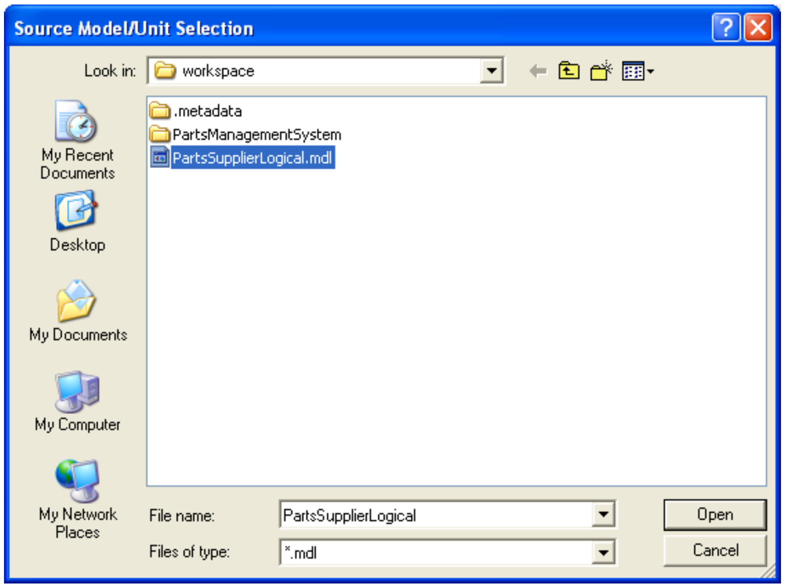
1. From the MetaBase Modeler menubar, select **File > Import**. The **Import** selection screen displays.


Import
Select
Import metadata from a IBM Rational Rose © model into a new or existing UML model.
Select an import source:
ERwin © Models
Existing Project into Workspace
File system
IBM Rational Rose © Model
Metadata from JDBC Database
MetaMatrix 3.1 Models
Team Project Set
Zip file
< Back
Next >
Finish
Cancel

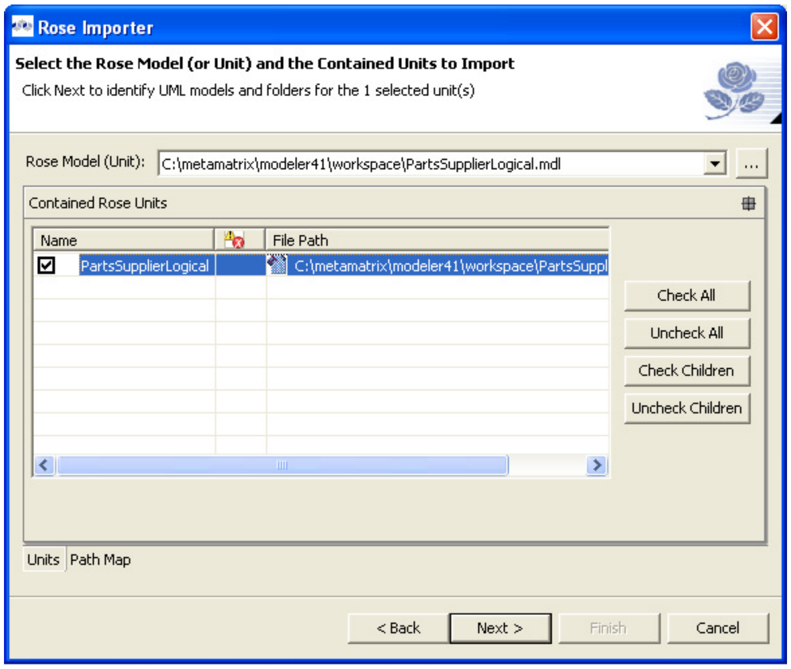
2. Select an **IBM Rational Rose Model** as your import source. Click **Next**. The **Rose Importer** dialog box displays.



Using the ellipse button ..., navigate to a Rose model unit you wish to import.



3. After you return to the main Rose Importer dialog, check the appropriate models and/or model children to import. Click **Next**.



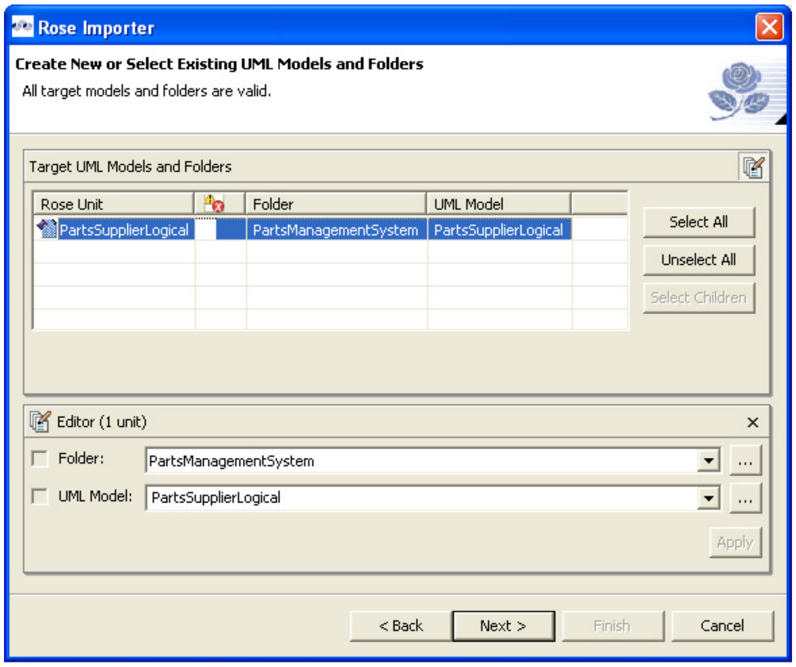
Select the Rose Model (or Unit) and the Contained Units to Import
Click Next to identify UML models and folders for the 1 selected unit(s)

Rose Model (Unit): C:\metamatrix\modeler41\workspace\PartsSupplierLogical.mdl

Contained Rose Units

Name		File Path
☑ PartsSupplierLogical		C:\metamatrix\modeler41\workspace\PartsSuppl

4. The **Create New or Select Existing UML** dialog appears.



Create New or Select Existing UML Models and Folders
All target models and folders are valid.

Target UML Models and Folders

Rose Unit		Folder	UML Model
PartsSupplierLogical		PartsManagementSystem	PartsSupplierLogical

Editor (1 unit)

Folder: PartsManagementSystem

UML Model: PartsSupplierLogical

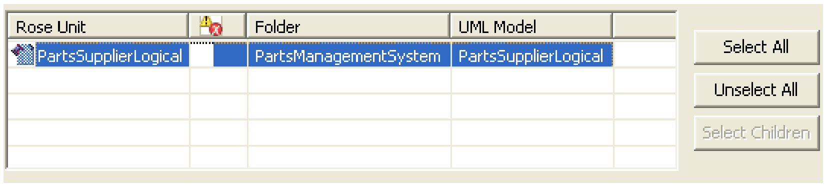
5. Select the target models and folders.

NOTE: If preserving the file system structure of the source Rose Units being imported is desired, a similar file system structure must exist in the model target location prior to starting the wizard.

This wizard page consists of two distinct areas separated by an adjustable, split-pane dialog box:

- The selected source Rose Units table
- The editor

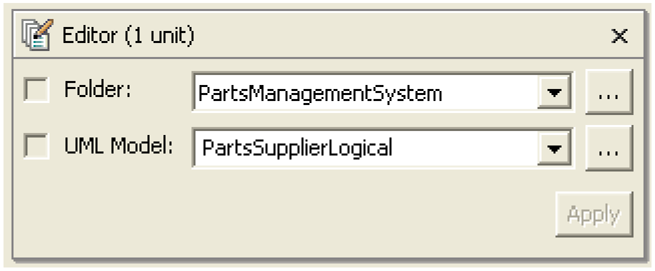
The top area consists of a table identifying the Rose Units being imported.



Rose Unit		Folder	UML Model
PartsSupplierLogical		PartsManagementSystem	PartsSupplierLogical

For each Rose Unit a target folder and target model name is shown, as well as, an error or warning indicator. The initial value of the target folder will be set to the model project or folder that was selected prior to starting the wizard (or none if there was no previous selection). Selection buttons to the right of the table aid in selecting table rows. Values in the selected table rows are changed via the editor.

The bottom area is an editor.



Editor (1 unit)

Folder: PartsManagementSystem

UML Model: PartsSupplierLogical

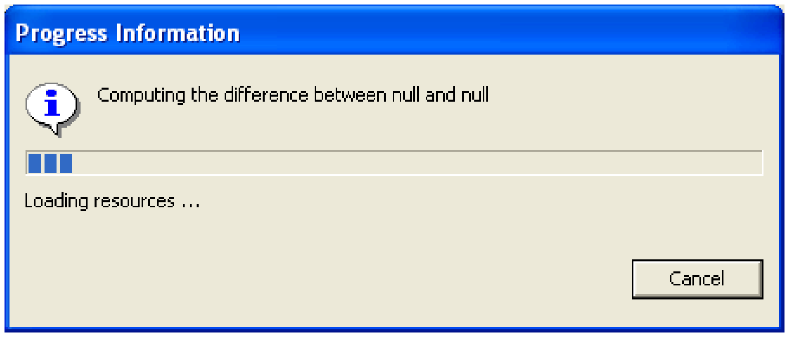
The editor allows target folders and model names to be changed. The editor modifies all selected table rows (the top area) with its information once the **Apply** button is selected. If more than one row is selected, the information area shows only the common information.

For example, if two rows are selected that have the same target folder, then that target folder will show in the editor. However, if two rows are selected with different target folders, the editor will not show any information for the folder.

When changes are made in the editor, the appropriate checkbox is automatically checked. Unchecking the checkbox will reset the editor back to the original value.

For both the target folder and target model properties, using their respective browse buttons can set new values. This allows navigation of the workspace. Selecting a recently used value in the drop-down dialog can also set new values. The editor can be closed and opened either by using the editor's toolbar button or the top areas toolbar button.

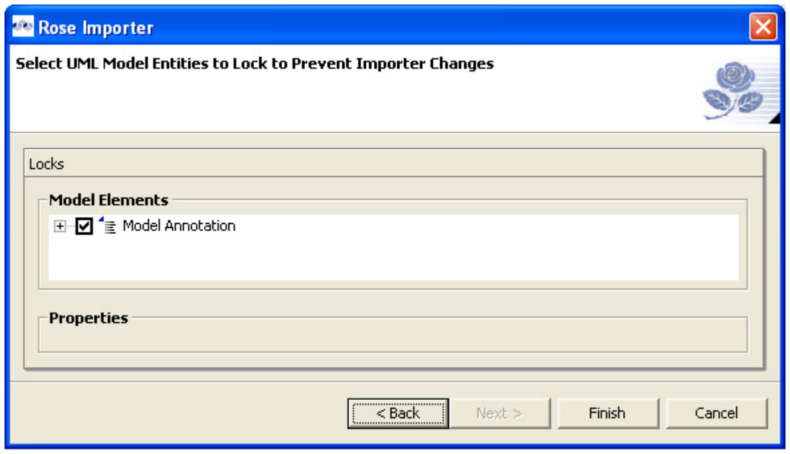
6. Click **Next**. A brief progress information bar appears.


Progress Information

Computing the difference between null and null

Loading resources ...

7. You are presented with a dialog giving you the opportunity to lock elements of the UML model entity(s).


Rose Importer

Select UML Model Entities to Lock to Prevent Importer Changes

Locks

Model Elements

☑ Model Annotation

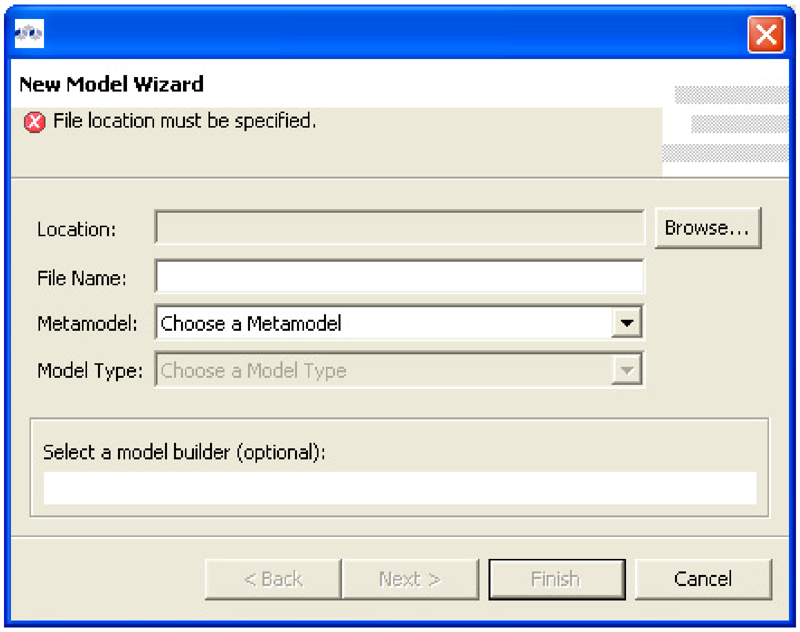
Properties

8. Click **Finish**.

Adding Relationships to UML Models

To create a secondary relationship model (working in conjunction with a primary UML model), use the following procedure.

1. From the MetaBase Modeler menu bar, select **File > New > Metadata Model**. The New Model Wizard dialog displays.



New Model Wizard

File location must be specified.

Location:

File Name:

Metamodel: Choose a Metamodel

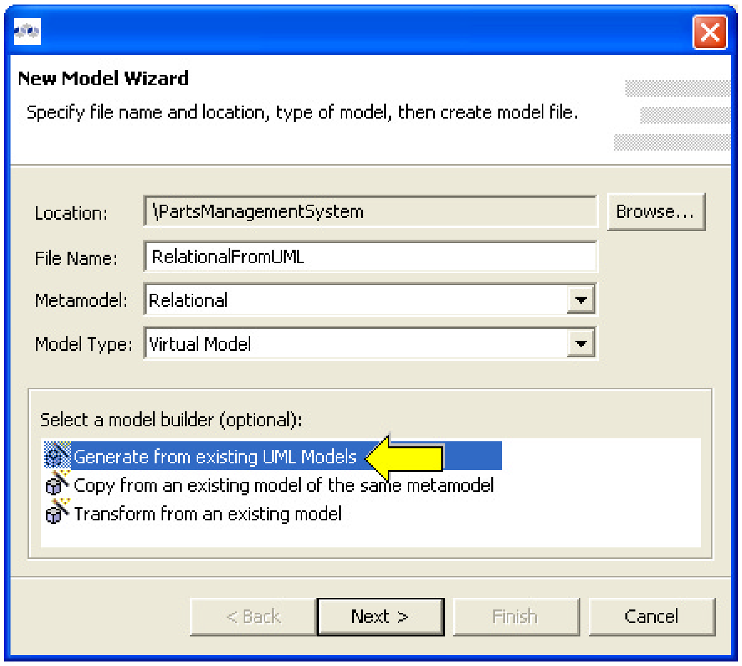
Model Type: Choose a Model Type

Select a model builder (optional):

2. Browse to the location that the model will be placed, give the file a relevant file name, and choose Relational from the **Metamodel:** drop down dialog.

   Relational models can be either virtual or physical.

3. Highlight **Generate from existing UML Models**.



New Model Wizard

Specify file name and location, type of model, then create model file.

Location: \PartsManagementSystem

File Name: RelationalFromUML

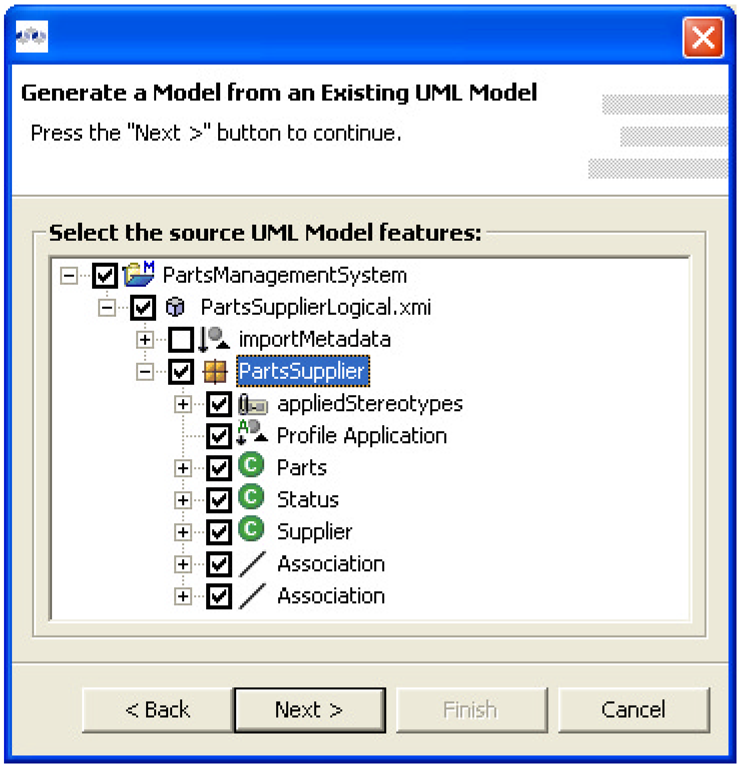
Metamodel: Relational

Model Type: Virtual Model

Select a model builder (optional):

- Generate from existing UML Models
- Copy from an existing model of the same metamodel
- Transform from an existing model

4. Click **Next**. The **Generate a Model from an Existing UML Model** dialog displays.



Generate a Model from an Existing UML Model

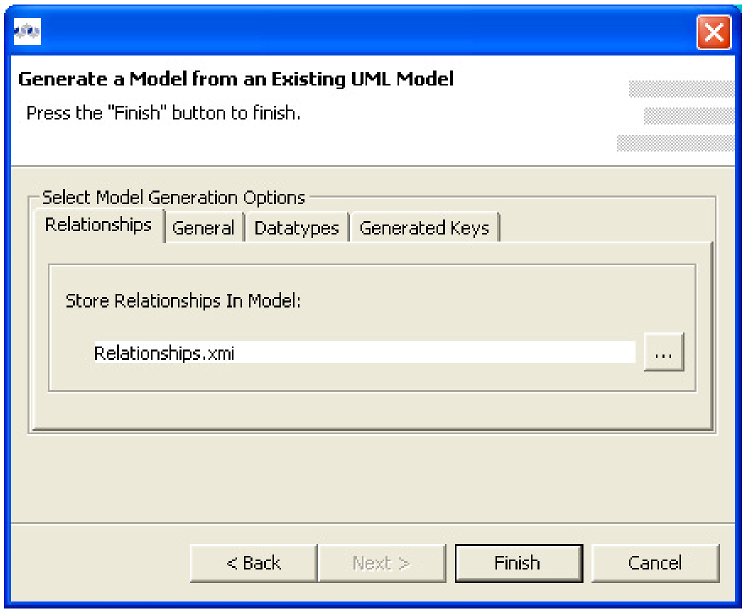
Press the "Next >" button to continue.

Select the source UML Model features:

- ✓ PartsManagementSystem
  - ✓ PartsSupplierLogical.xmi
    - ☐ importMetadata
    - ✓ PartsSupplier
      - ✓ appliedStereotypes
      - ✓ Profile Application
      - ✓ Parts
      - ✓ Status
      - ✓ Supplier
      - ✓ Association
      - ✓ Association

5. Select the Source UML entities that will generate the Relational Model.

6. Select the Relationships model in which all of the generated relationships will be placed.

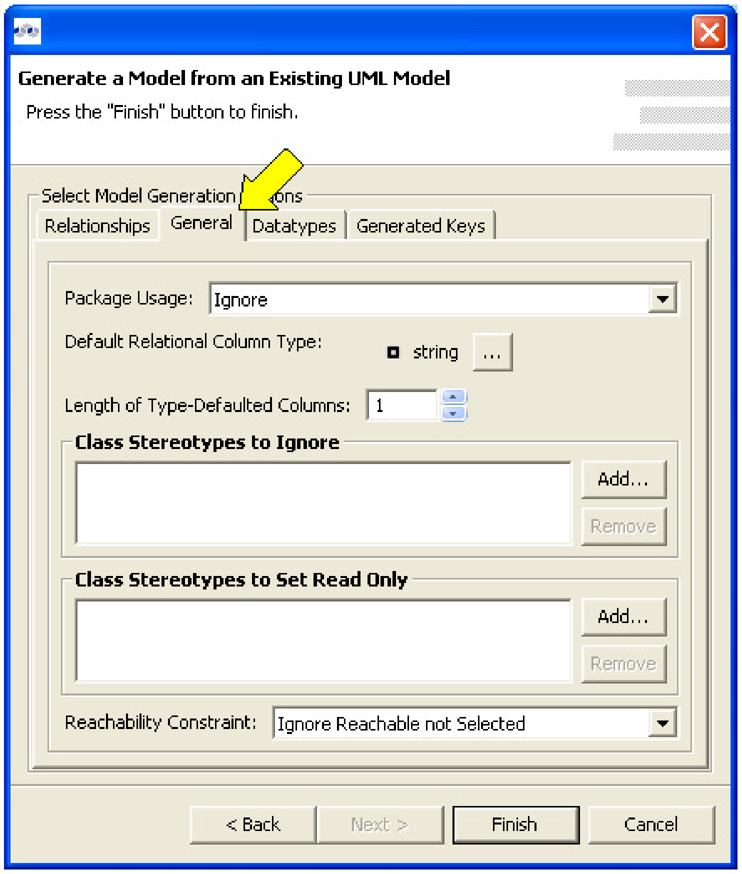


Generate a Model from an Existing UML Model
Press the "Finish" button to finish.

Select Model Generation Options

Store Relationships In Model:
Relationships.xmi

7. The **General** options tab contains more optional parameters for the relationship model.



Generate a Model from an Existing UML Model
Press the "Finish" button to finish.

Select Model Generation Options

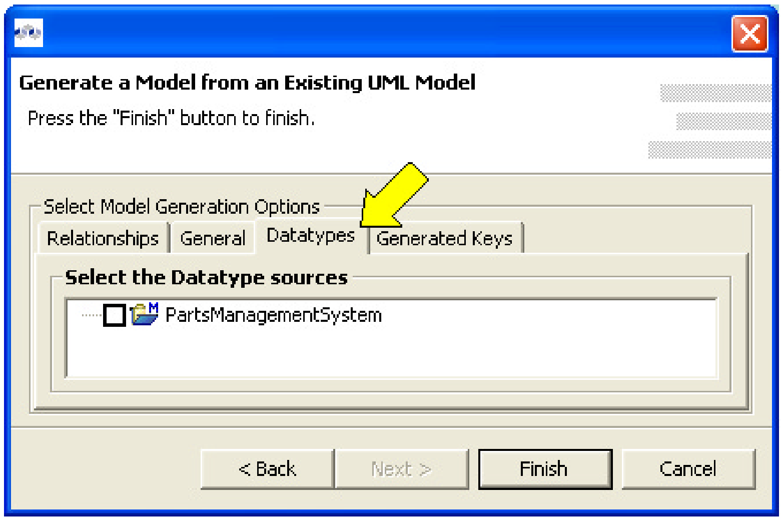
Package Usage: Ignore
Default Relational Column Type: string
Length of Type-Defaulted Columns: 1

Class Stereotypes to Ignore

Class Stereotypes to Set Read Only

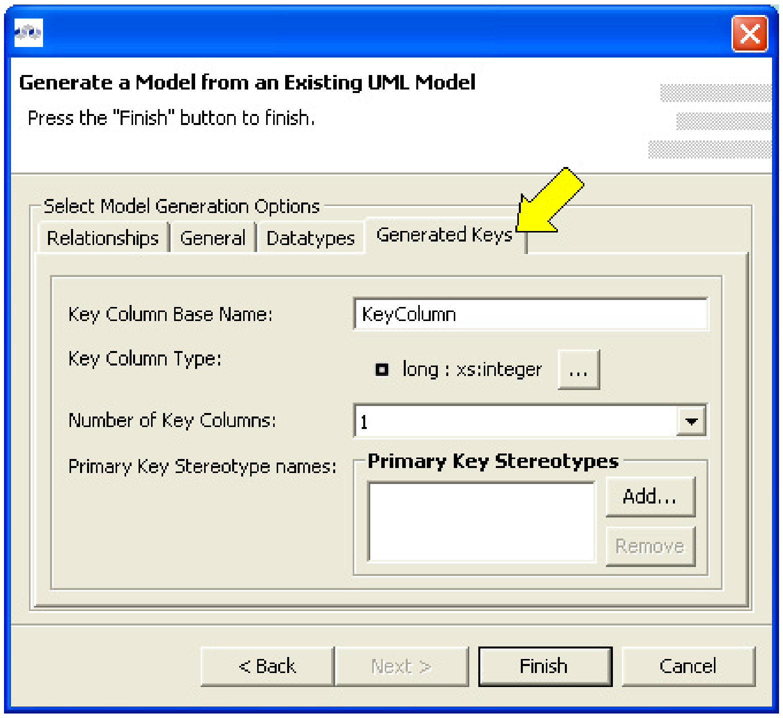
Reachability Constraint: Ignore Reachable not Selected

8. The **Datatypes** tab contains more optional parameters for the relationship model.



Generate a Model from an Existing UML Model
Press the "Finish" button to finish.
Select Model Generation Options
Relationships
General
Datatypes
Generated Keys
Select the Datatype sources
PartsManagementSystem
< Back
Next >
Finish
Cancel

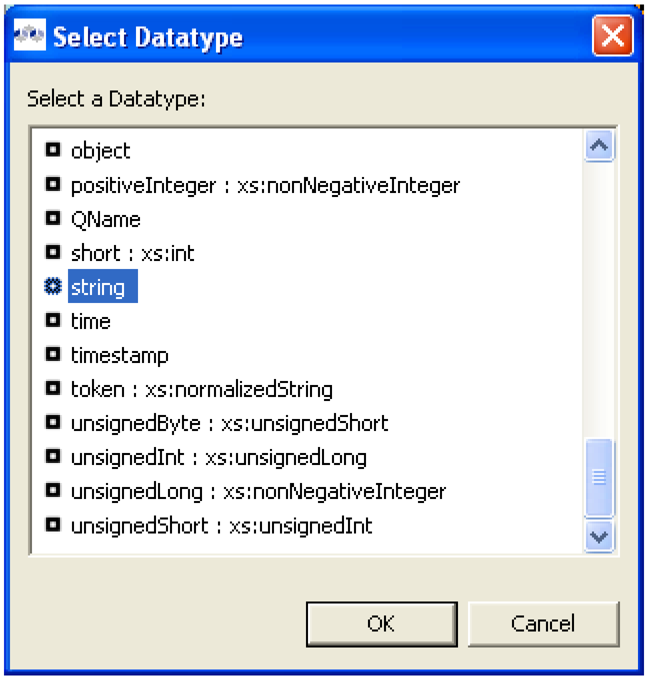
9. The **Generated Keys** tab contains more optional parameters for the relationship model.



Generate a Model from an Existing UML Model
Press the "Finish" button to finish.
Select Model Generation Options
Relationships
General
Datatypes
Generated Keys
Primary Key Stereotypes
Add...
Remove
< Back
Next >
Finish
Cancel

Key Column Base Name: KeyColumn
Key Column Type: long : xs:integer
Number of Key Columns: 1
Primary Key Stereotype names:

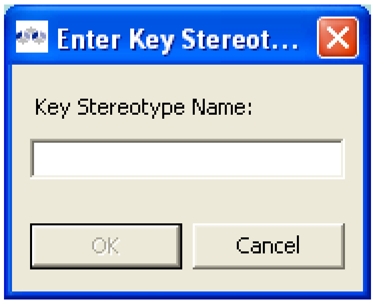
10. The ellipse button ... launches a **Select Datatype** menu.



Select a Datatype:

- object
- positiveInteger : xs:nonNegativeInteger
- QName
- short : xs:int
- string
- time
- timestamp
- token : xs:normalizedString
- unsignedByte : xs:unsignedShort
- unsignedInt : xs:unsignedLong
- unsignedLong : xs:nonNegativeInteger
- unsignedShort : xs:unsignedInt

11. The **Add Primary Key** Stereotypes button launches an entry dialog box for the **Key Stereotype Name**.



CONNECTION INFORMATION IN METADATA MODELS

Connection-Related Meta Objects

The Metadata Import Wizards provided by MetaMatrix store information describing the connection used by the Metadata Import Wizard. You will find these meta objects in physical model, and they describe the enterprise information system from which you imported the model. You cannot use this information in virtual metadata models or transformations based upon the physical metadata model.

Refreshing Metadata

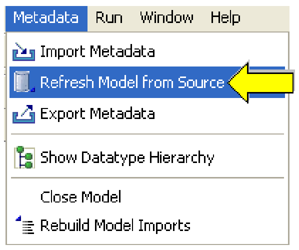
You can use this connection information to refresh the contents of certain meta objects and models to ensure the metadata remains consistent with structure of the data source it describes.

Refreshing differs from importing metadata into an existing model in that refreshing automatically uses the information in the connection meta objects instead of presenting the Metadata Import Wizard for you to choose the options again and refreshing only compares the meta objects in your existing model, whereas updating a model allows you to broaden the scope to include other meta objects.

You can refresh the following:

- Relational models.
- Data access models.

To refresh the model, select the model you want to refresh on the **Model Explorer** or **Outline** view and select **Metadata > Refresh Model from Source.**



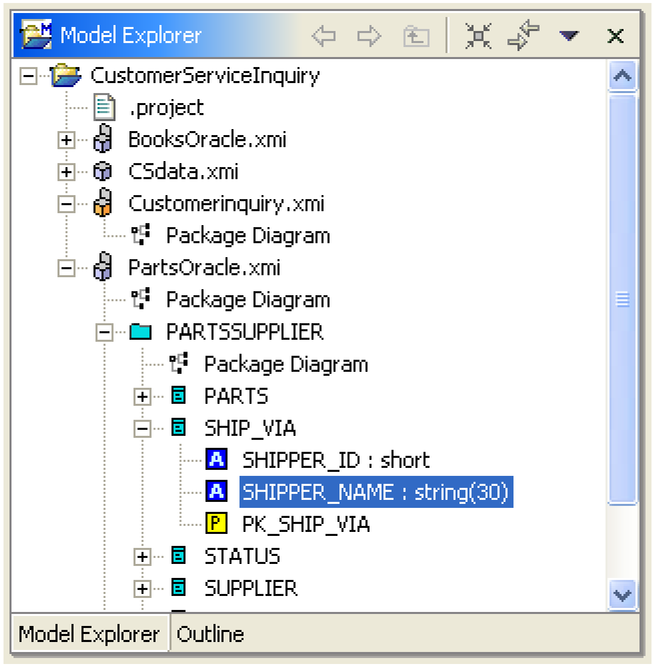
The MetaBase Modeler will use the connection information in the stored with the model and the Metadata Import Wizard plug-in named in the **Connections** meta object to automatically re-import the metadata. Keep in mind you need the plug-in named in the Connections meta object installed on your workstation to refresh metadata from the source.

Chapter 7: Navigating Metadata

NAVIGATING THE MODEL/EXPLORER TREE

The **Model Explorer** view contains a hierarchical organization of the metadata within the project.

Once you have opened a model, your **Model Explorer** view looks something like this:



```
CustomerServiceInquiry
    .project
    BooksOracle.xmi
    CSdata.xmi
    Customerinquiry.xmi
        Package Diagram
    PartsOracle.xmi
        Package Diagram
        PARTSSUPPLIER
            Package Diagram
            PARTS
            SHIP_VIA
                SHIPPER_ID : short
                SHIPPER_NAME : string(30)
                PK_SHIP_VIA
            STATUS
            SUPPLIER
```

The toolbar at the top of the **Model Explorer** view offers a set of buttons to help you navigate the project files.

Buttons in the Model Explorer View

The toolbar at the top of the **Model Explorer view** offers you a quick way to navigate the models you have open in your workspace.

Button	Function
⬅	Select the last item you viewed before the current node.
➡	Select the next item you selected after the current node.
⮤	Select the parent meta object of the currently selected node.
⤡	Collapse all nodes.
⇄	Link with Editor. When toggled "on" it synchronizes any actions done in the Editor Panel view or Description view with an immediate refresh of the Model Explorer view.
⇉	Filters. You can specify which kinds of files display or do not display in the Model Explorer view.

Icons in the Model Explorer View

The icon beside the node of the tree tells you what sort of meta object the node represents.

Icon	Meta Object	Examples (if applicable)
📦	Physical metadata model	
📦	Virtual metadata model	
📁	Package	Category (Data Access metamodel) Schema (Relational metamodel) Catalog (Relational metamodel)
📁	Virtual package	Any of the above packages in a Virtual metadata model
▤	Class	Group (Data Access metamodel) Base Table (Relational metamodel) View (Relational metamodel)
▤	Class	Any of the above classes in a Virtual metadata model
A	Attribute	Element (Data Access metamodel) Column (Relational metamodel)
K	Unique Constraint	
P	Primary Key	
F	Foreign Key	
▯	JDBC Sources	
▶	JDBC Import Settings	
I	Index	
▦	Diagram	Package Custom
▦	Transformation Diagram	
▦	Mapping Diagram	
▤	Document	XML Schema (XML Schema metamodel) XML document (XML Document metamodel)
{!}	Comment	Documentation (XML Schema metamodel) Comment (XML Document metamodel)
{ }	Compositor	All Compositor (XML Schema metamodel) Sequence Compositor (XML Schema metamodel)
⚙	Atomic datatype	
●	Complex datatype	
{a}	Attribute	
{e}	Element	
{:}	Namespace	
{i}	Import	Import (XML Schema metamodel) Include (XML Schema metamodel)
▸	Pattern, Enumeration	
●	Connection	

Click any node in the **Model Explorer/Outline** view and the properties will display in the **Properties** and **Description** panel. You can modify many properties in the **Description** view if you can write to the model in your local directory (you have not checked it into the MetaBase Repository).

Chapter 8: Creating User-Defined Datatypes

WHAT ARE DATATYPES?

Datatypes represent what sort of information a meta object contains. For example, does it contain:

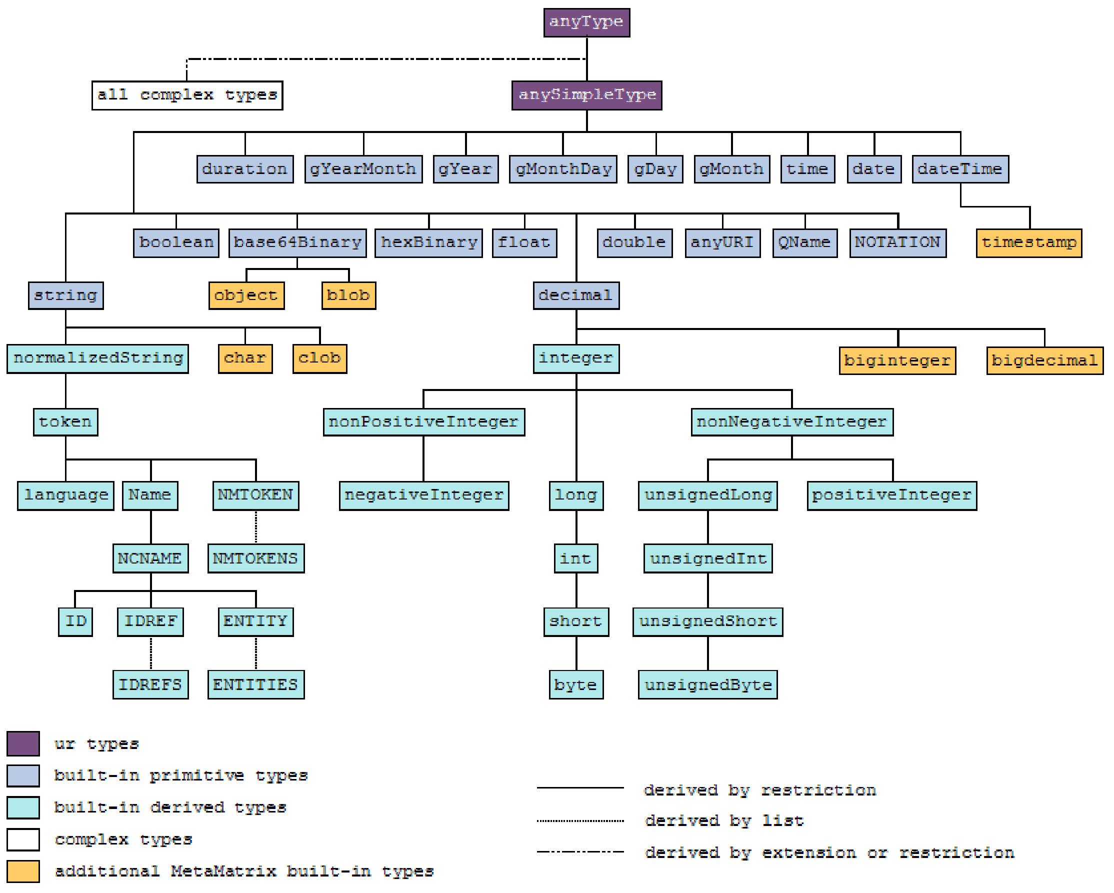
- A number (an integer)
- A number with a decimal point (a float)
- A yes-or-no value, sometimes called a flag (a Boolean)

The datatype represents the type of variable or parameter used to store that information within the data source.

When you create derived datatypes in the MetaBase Modeler, you're essentially creating a metadata model using the special classes and attributes that reflect your new datatypes.

BUILT-IN DATATYPES

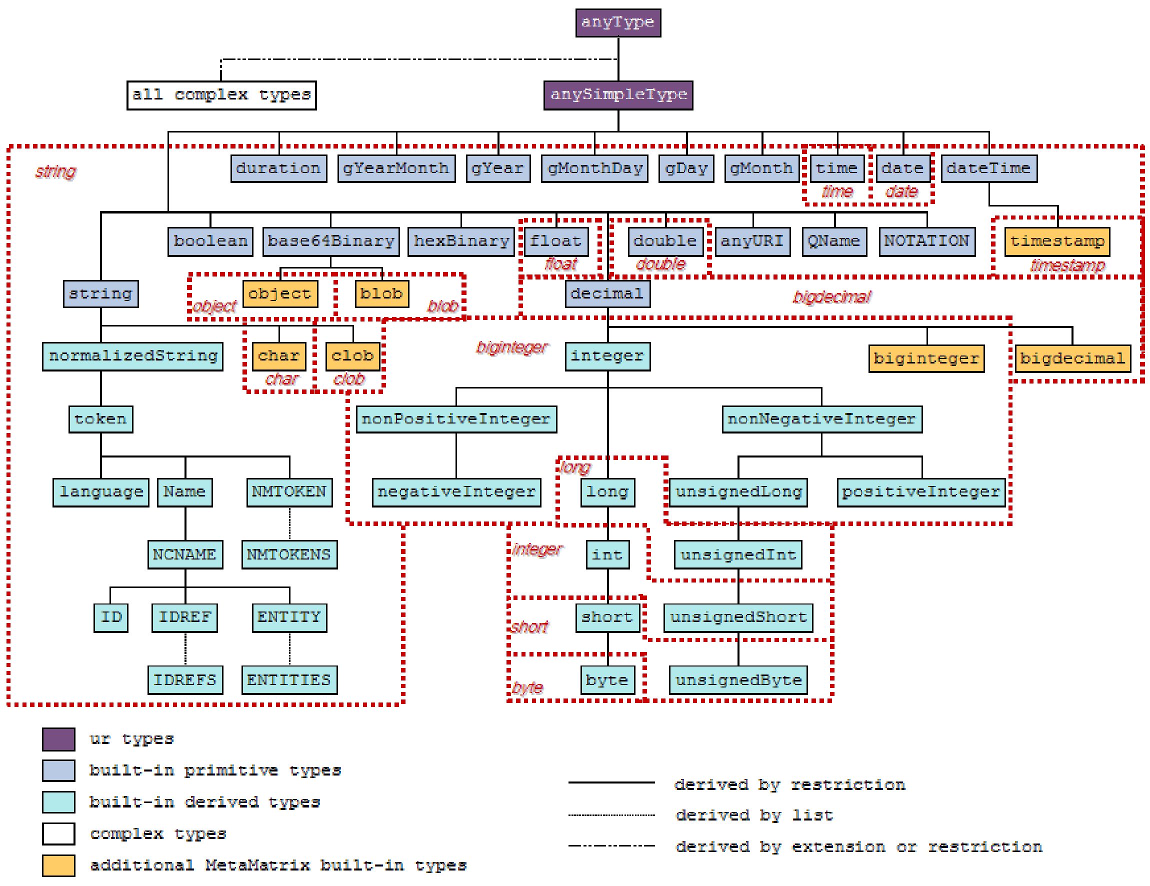
The MetaBase Modeler offers a wide variety of common datatypes to accommodate common data source datatypes. In addition to reusing all of the XML Schema built-in types, MetaMatrix offers seven additional types. See the diagram that follows for a complete accounting.



BUILT-IN DATATYPES WITH RUNTIME TYPES

MetaMatrix associates a **runtime type** with each datatype (including built-in datatypes). This runtime type defines how the MetaMatrix Server works with the values of that type during query execution.

The following image shows built-in datatypes and their associated runtime times defining how enterprise information source data is accessed and manipulated using the MetaMatrix Server.



DATATYPE DEFINITIONS

Following are samples and definitions for MetaMatrix built-in datatypes referenced in the two previous illustrations.

anyURI

Runtime Datatype

java.lang.String

Definition

anyURI represents a Uniform Resource Identifier Reference (URI). An anyURI value can be absolute or relative, and may have an optional fragment identifier (i.e., it may be a URI Reference). This type should be used to specify the intention that the value fulfills the role of a URI as defined by [RFC 2396], as amended by [RFC 2732].

base64Binary

Runtime Datatype

java.lang.String

Definition

base64Binary represents Base64-encoded arbitrary binary data. The value space of base64Binary is the set of finite-length sequences of binary octets. For base64Binary data the entire binary stream is encoded using the Base64 Content-Transfer-Encoding defined in Section 6.8 of [RFC 2045].

bigdecimal

Runtime Datatype

java.math.BigDecimal

Definition

bigdecimal represents arbitrary-precision signed decimal numbers. A bigdecimal consists of an arbitrary precision integer unscaled value and a non-negative 32-bit integer scale, which represents the number of digits to the right of the decimal point. The number represented by the bigdecimal is (unscaledValue/10scale).

Biginteger

Runtime Datatype

java.math.BigInteger

Definition

biginteger represent arbitrary-precision integers. All operations behave as if bigintegers were represented in two's-complement notation (like Java's primitive integer types).

blob

Runtime Datatype

com.metamatrix.common.types.BlobType

Definition

blob represents a binary large object.

boolean

Runtime Datatype

java.lang.Boolean

Definition

boolean has the value space required to support the mathematical concept of binary-valued logic: {true, false}.

byte

Runtime Datatype

java.lang.Byte

Definition

byte is derived from short by setting the value of maxInclusive to be 127 and minInclusive to be -128. The base type of byte is short.

char

Runtime Datatype

java.lang.Character

Definition

char represents a single character data type.

clob

Runtime Datatype

com.metamatrix.common.types.ClobType

Definition

clob represents a character large object.

date

Runtime Datatype

java.sql.Date

Definition

date represents a calendar date. The value space of date is the set of Gregorian calendar dates as defined in 5.2.1 of [ISO 8601]. Specifically, it is a set of one-day long, non-periodic instances e.g. lexical 1999-10-26 to represent the calendar date 1999-10-26, independent of how many hours this day has.

dateTime

Runtime Datatype

java.lang.String

Definition

dateTime represents a specific instant of time. The value space of dateTime is the space of Combinations of date and time of day values as defined in 5.4 of [ISO 8601].

decimal

Runtime Datatype

java.math.BigDecimal

Definition

decimal represents arbitrary precision decimal numbers. The value space of decimal is the set of the values **i** - **n**, where **i** and **n** are integers such that **n** >= 0.

The order-relation on decimal is:

x < **y** if **y** - **x** is positive. The value space of types derived from decimal with a value for totalDigits of **p** is the set of values **i** - **n**, where **n** and **i** are integers such that **p** >= **n** >= 0 and the number of significant decimal digits in **i** is less than or equal to **p**. The value space of types derived from decimal with a value for fractionDigits of s is the set of values **i** - **n**, where **i** and **n** are integers such that 0 <= **n** <= **s**.

double

Runtime Datatype

java.lang.Double

Definition

The **double** datatype corresponds to IEEE double-precision 64-bit floating point type [IEEE 754-1985]. The basic value space of double consists of the values **m**, where **m** is an integer whose absolute value is less than 2^53, and **e** is an integer between -1075 and 970, inclusive. In addition to the basic value space described above, the value space of double also contains the following special values: positive and negative zero, positive and negative infinity and not-a-number.

The order-relation on double is: **x** < **y** if **y** - **x** is positive. Positive zero is greater than negative zero. Not-a-number equals itself and is greater than all double values including positive infinity.

duration

Definition

duration represents a duration of time. The value space of duration is a six-dimensional space where the coordinates designate the Gregorian year, month, day, hour, minute, and second components defined in 5.5.3.2 of [ISO 8601], respectively. These components are ordered in their significance by their order of appearance i.e. as year, month, day, hour, minute, and second.

ENTITIES

Runtime Datatype

java.lang.String

Definition

ENTITIES represents the ENTITIES attribute type from [XML 1.0 (Second Edition)]. The value space of ENTITIES is the set of finite, non-zero-length sequences of ENTITYs that have been declared as unparsed entities in a document type definition. The lexical space of ENTITIES is the set of white space separated lists of tokens, of which each token is in the lexical space of ENTITY. The itemType of ENTITIES is ENTITY.

ENTITY

Runtime Datatype

java.lang.String

Definition

ENTITY represents the ENTITY attribute type from [XML 1.0 (Second Edition)]. The value space of ENTITY is the set of all strings that match the NCName production in [Namespaces in XML] and have been declared as an unparsed entity in a document type definition. The lexical space of ENTITY is the set of all strings that match the NCName production in [Namespaces in XML]. The base type of ENTITY is NCName.

float

Runtime Datatype

java.lang.Float

Definition

float corresponds to the IEEE single-precision 32-bit floating point type [IEEE 754-1985]. The basic value space of float consists of the values m, where m is an integer whose absolute value is less than 2^24, and e is an integer between -149 and 104, inclusive. In addition to the basic value space described above, the value space of float also contains the following special values: positive and negative zero, positive and negative infinity and not-a-number. The order-relation on float is: **x** < **y** if **y** - **x** is positive. Positive zero is greater than negative zero. Not-a-number equals itself and is greater than all float values including positive infinity.

gDay

Runtime Datatype

java.lang.String

Definition

gDay is a Gregorian day that recurs, specifically a day of the month such as the 5th of the month. Arbitrary recurring days are not supported by this datatype. The value space of gDay is the space of a set of calendar dates as defined in 3 of [ISO 8601]. Specifically, it is a set of one-day long, monthly periodic instances.

gMonth

Runtime Datatype

java.lang.String

Definition

gMonth is a Gregorian month that recurs every year. The value space of gMonth is the space of a set of calendar months as defined in 3 of [ISO 8601]. Specifically, it is a set of one-month long, yearly periodic instances.

gMonthDay

Runtime Datatype

java.lang.String

Definition

gMonthDay is a Gregorian date that recurs, specifically a day of the year such as the third of May. Arbitrary recurring dates are not supported by this datatype. The value space of gMonthDay is the set of calendar dates, as defined in 3 of [ISO 8601]. Specifically, it is a set of one-day long, annually periodic instances.

gYear

Runtime Datatype

java.lang.String

Definition

gYear represents a Gregorian calendar year. The value space of gYear is the set of Gregorian calendar years as defined in 5.2.1 of [ISO 8601]. Specifically, it is a set of one-year long, non-periodic instances (e.g. lexical 1999 to represent the whole year 1999), independent of how many months and days this year has.

gYearMonth

Runtime Datatype

java.lang.String

Definition

gYearMonth represents a specific Gregorian month in a specific Gregorian year. The value space of gYearMonth is the set of Gregorian calendar months as defined in 5.2.1 of [ISO 8601]. Specifically, it is a set of one-month long, non-periodic instances e.g. 1999-10 to represent the whole month of 1999-10, independent of how many days this month has.

hexBinary

Runtime Datatype

java.lang.String

Definition

hexBinary represents arbitrary hex-encoded binary data. The value space of hexBinary is the set of finite-length sequences of binary octets.

ID

Runtime Datatype

java.lang.String

Definition

ID represents the ID attribute type from [XML 1.0 (Second Edition)]. The value space of ID is the set of all strings that match the NCName production in [Namespaces in XML]. The lexical space of ID is the set of all strings that match the NCName production in [Namespaces in XML]. The base type of ID is NCName.

IDREF

Runtime Datatype

java.lang.String

Definition

IDREF represents the IDREF attribute type from [XML 1.0 (Second Edition)]. The value space of IDREF is the set of all strings that match the NCName production in [Namespaces in XML]. The lexical space of IDREF is the set of strings that match the NCName production in [Namespaces in XML]. The base type of IDREF is NCName.

IDREFS

Runtime Datatype

java.lang.String

Definition

IDREFS represents the IDREFS attribute type from [XML 1.0 (Second Edition)]. The value space of IDREFS is the set of finite, non-zero-length sequences of IDREFs. The lexical space of IDREFS is the set of white space separated lists of tokens, of which each token is in the lexical space of IDREF. The itemType of IDREFS is IDREF.

int

Runtime Datatype

java.lang.Integer

Definition

int is derived from long by setting the value of maxInclusive to be 2147483647 and minInclusive to be -2147483648. The base type of int is long.

integer

Runtime Datatype

java.math.BigInteger

Definition

integer is derived from decimal by fixing the value of fractionDigits to be 0. This results in the standard mathematical concept of the integer numbers. The value space of integer is the infinite set {...,-2,-1,0,1,2,...}. The base type of integer is decimal.

language

Runtime Datatype

java.lang.String

Definition

language represents natural language identifiers as defined by [RFC 1766]. The value space of language is the set of all strings that are valid language identifiers as defined in the language identification section of [XML 1.0 (Second Edition)]. The lexical space of language is the set of all strings that are valid language identifiers as defined in the language identification section of [XML 1.0 (Second Edition)]. The base type of language is token.

long

Runtime Datatype

java.lang.Long

Definition

long is derived from integer by setting the value of maxInclusive to be 9223372036854775807 and minInclusive to be -9223372036854775808. The base type of long is integer.

Name

Runtime Datatype

java.lang.String

Definition

Name represents XML Names. The value space of Name is the set of all strings which match the Name production of [XML 1.0 (Second Edition)]. The lexical space of Name is the set of all strings that match the Name production of [XML 1.0 (Second Edition)]. The base type of Name is token.

NCName

Runtime Datatype

java.lang.String

Definition

NCName represents ML"non-colonized" Names. The value space of NCName is the set of all strings which match the NCName production of [Namespaces in XML]. The lexical space of NCName is the set of all strings that match the NCName production of [Namespaces in XML]. The base type of NCName is Name.

negativeInteger

Runtime Datatype

java.math.BigInteger

Definition

negativeInteger is derived from nonPositiveInteger by setting the value of maxInclusive to be -1. This results in the standard mathematical concept of the negative integers. The value space of negativeInteger is the infinite set {...,-2,-1}. The base type of negativeInteger is nonPositiveInteger.

NMTOKEN

Runtime Datatype

java.lang.String

Definition

NMTOKEN represents the NMTOKEN attribute type from [XML 1.0 (Second Edition)]. The value space of NMTOKEN is the set of tokens that match the Nmtoken production in [XML 1.0 (Second Edition)]. The lexical space of NMTOKEN is the set of strings that match the Nmtoken production in [XML 1.0 (Second Edition)]. The base type of NMTOKEN is token.

NMTOKENS

Runtime Datatype

java.lang.String

Definition

NMTOKENS represents the NMTOKENS attribute type from [XML 1.0 (Second Edition)].The value space of NMTOKENS is the set of finite, non-zero-length sequences of NMTOKENs. The lexical space of NMTOKENS is the set of white space separated lists of tokens, of which each token is in the lexical space of NMTOKEN. The itemType of NMTOKENS is NMTOKEN.

nonNegativeInteger

Runtime Datatype

java.math.BigInteger

Definition

nonNegativeInteger is derived from integer by setting the value of minInclusive to be 0. This results in the standard mathematical concept of the non-negative integers. The value space of nonNegativeInteger is the infinite set {0,1,2,...}. The base type of nonNegativeInteger is integer.

nonPositiveInteger

Runtime Datatype

java.math.BigInteger

Definition

nonPositiveInteger is derived from integer by setting the value of maxInclusive to be 0. This results in the standard mathematical concept of the non-positive integers. The value space of nonPositiveInteger is the infinite set {...,-2,-1,0}. The base type of nonPositiveInteger is integer.

normalizedString

Runtime Datatype

java.lang.String

Definition

normalizedString represents white space normalized strings. The value space of normalizedString is the set of strings that do not contain the carriage return (#xD), line feed (#xA) nor tab (#x9) characters. The lexical space of normalizedString is the set of strings that do not contain the carriage return (#xD) nor tab (#x9) characters. The base type of normalizedString is string.

NOTATION

Runtime Datatype

java.lang.String

Definition

NOTATION represents the NOTATION attribute type from [XML 1.0 (Second Edition)]. The value space of NOTATION is the set QNames. The lexical space of NOTATION is the set of all names of notations declared in the current schema. NOTATION cannot be used directly in a model; rather a type must be derived from it by specifying at least one enumeration facet whose value is the name of a NOTATION declared in the model.

object

Runtime Datatype

java.lang.Object

Definition

object represents a java.lang.Object

positiveInteger

Runtime Datatype

java.math.BigInteger

Definition

positiveInteger is derived from nonNegativeInteger by setting the value ofminInclusive to be 1. This results in the standard mathematical concept of the positive integer numbers. The value space of positiveInteger is the infinite set {1,2,...}. The base type of positiveInteger is nonNegativeInteger.

QName

Runtime Datatype

java.lang.String

Definition QName represents XML qualified names. The value space of QName is the set of tuples {namespace name, local part}, where namespace name is an anyURI and local part is an NCName. The lexical space of QName is the set of strings that match the QName production of [Namespaces in XML].

short

Runtime Datatype

java.lang.Short

Definition

short is derived from **int** by setting the value of maxInclusive to be 32767 and minInclusive to be -32768. The base type of short is int.

string

Runtime Datatype

java.lang.String

Definition

The string datatype represents character strings in XML. The value space of string is the set of finite-length sequences of characters (as defined in [XML 1.0 (Second Edition)]) that match the Char production from [XML 1.0 (Second Edition)]. A character is an atomic unit of communication; it is not further specified except to note that every character has a corresponding Universal Character Set code point, which is an integer.

time

Runtime Datatype

java.sql.Time

Definition

time represents an instant of time that recurs every day. The value space of time is the space of time of day values as defined in 5.3 of [ISO 8601]. Specifically, it is a set of zero-duration daily time instances.

timestamp

Runtime Datatype

java.sql.Timestamp

Definition

timestamp represents date that allows the JDBC API to identify this as an SQL TIMESTAMP value. It adds the ability to hold the SQL TIMESTAMP nanos value and provides formatting and parsing operations to support the JDBC escape syntax for timestamp values. Note: This type is a composite of a java.util.Date and a separate nanoseconds value. Only integral seconds are stored in the java.util.Date component. The fractional seconds - the nanos - are separate. The standard ANSI SQL timestamp format is YYYY-MM-DD HH:MM:SS[.xxxxxxxxx] Hours are 24-hour time. The nanoseconds are optional.

token

Runtime Datatype

java.lang.String

Definition

token represents tokenized strings. The value space of token is the set of strings that do not contain the line feed (#xA) nor tab (#x9) characters, that have no leading or trailing spaces (#x20) and that have no internal sequences of two or more spaces. The lexical space of token is the set of strings that do not contain the line feed (#xA) nor tab (#x9) characters, that have no leading or trailing spaces (#x20) and that have no internal sequences of two or more spaces. The base type of token is normalizedString.

unsignedByte

Runtime Datatype

java.lang.Short

Definition

unsignedByte is derived from unsignedShort by setting the value of maxInclusive to be 255. The base type of unsignedByte is unsignedShort.

unsignedInt

Runtime Datatype

java.lang.Long

Definition

unsignedInt is derived from unsignedLong by setting the value of maxInclusive to be 4294967295. The base type of unsignedInt is unsignedLong.

unsignedLong

Runtime Datatype

java.math.BigInteger

Definition

unsignedLong is derived from nonNegativeInteger by setting the value of maxInclusive to be 18446744073709551615. The base type of unsignedLong is nonNegativeInteger.

unsignedShort

Runtime Datatype

java.lang.Integer

Definition

unsignedShort is derived from unsignedInt by setting the value of maxInclusive to be 65535. The base type of unsignedShort is unsignedInt.

USER-DEFINED DATATYPES

You can create fully functioning models for publication to the SearchBase or for creation of runtime metadata with the MetaMatrix Server using only the built-in datatypes. In fact, any models that your organization created prior to version 3.0 of the MetaMatrix System use these types exclusively.

However, there may be occasions when the built-in datatypes are not a good match (e.g. different lengths, minimum lengths, patterns, etc.). In these instances you can create new datatypes, selecting the most appropriate built-in or custom type of base type.

The MetaMatrix MetaBase Modeler lets you derive your own user-defined datatypes to extend or restrict the built-in datatypes. When you create your own datatype, you must define a runtime type associated with it. Your organization can create its own datatypes for use in models you publish to the SearchBase or use to create runtime metadata.

WHY CREATE USER-DEFINED DATATYPES?

Because your organization can model its information sources using only the basic, built-in datatypes, modeling user-defined datatypes might seem an extraneous step in creating metadata models to describe your information systems and your data consumption. However, creating your own user-defined datatypes offers your organization many benefits.

Formalizing a Data Dictionary

When you create user-defined datatypes, you can use them enterprise-wide to describe information more distinctly. You can create datatypes that describe the nature of the information more completely than the existing datatypes.

For example, when confronted with the ZIPCode column within the Address Book database, you can model this information easily as a string or an integer; however, if your organization creates a derived datatype called “ZIPCodeDT,” you and other data modelers within this organization can use this new datatype specifically to model ZIP codes.

Describing Data Rules in Detail

By creating a custom user-defined datatypes, you can easily create rules that apply to information of that datatype. You can set allowable values for that datatype by:

- **Creating a pattern.**
  The pattern, a rule, describes the format of the data that the datatype can contain. For example, for the ZIPCodeDT, you could set the allowable values to include 5 digits, or 9 digits, or 5 digits followed by a hyphen and then 4 more digits.

- **Enumerating actual values.**
  Your datatype definition can include a list of actual values for the datatype. For example, you could create a datatype called ZIPCodeStL to specify ZIP Codes in St. Louis, Missouri, and establish that the allowable values for this datatype include 63043, 63141, 63104, and whatever other values instances of this datatype can contain.

Reusing Datatypes

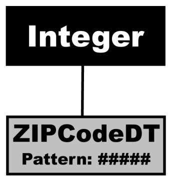
Once you have created a user-defined datatype, you can reuse that definition throughout your metadata models and in different metamodels. For example, you can not only model the ZIPCode column from the Address Relational database, using the Relational metamodel, but you can also use the ZIPCodeDT to model information within your XML data sources and others.

CREATING USER-DEFINED DATATYPES

When you model user-defined datatypes, you base your new datatype upon existing built-in datatypes or other user-defined datatypes. This ensures that you can use information modeled using your user-defined datatypes within your runtime metadata if you're using the MetaMatrix Server for data access.

Deriving from Built-In Datatypes

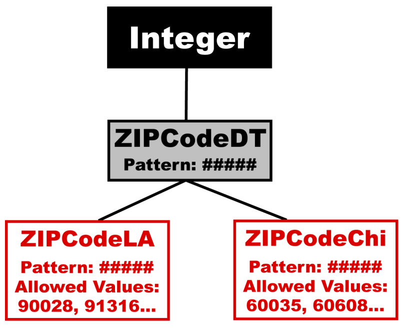
The most basic datatype user-defined derive directly from the built-in datatypes. For example, the ZIPCodeDT datatype relates directly to the integer built-in datatype. As such, it bears most of the characteristics of the integer datatype, but extends or limits the integer to a specific purpose or content:



This new ZIPCodeDT represents an integer that has the pattern of having five numbers in it. When you model a column as a ZIPCodeDT, it has all the characteristics of an integer but it only allow values comprised of five-digit numbers.

Deriving from Other User-Defined Datatypes

Once you have created user-defined datatypes, you can further extend or limit those datatypes according to your need. Again, your new user-defined datatypes bear the characteristics of the parent datatype and ultimately the characteristics of the base datatype:



The ZIPCodeLA and ZIPCodeChi datatypes both have the same patterns as their parent datatype, ZIPCodeDT, but each limits the allowed values, by enumeration, to certain literal values. Ultimately, both share characteristics of the built-in integer datatype.

MODELING THE USER-DEFINED DATATYPES

Within the MetaBase Modeler, you define your user-defined datatypes in a metadata model. You can then include this metadata model in projects with your other physical or virtual metadata models to use your user-defined datatypes within those metadata models.

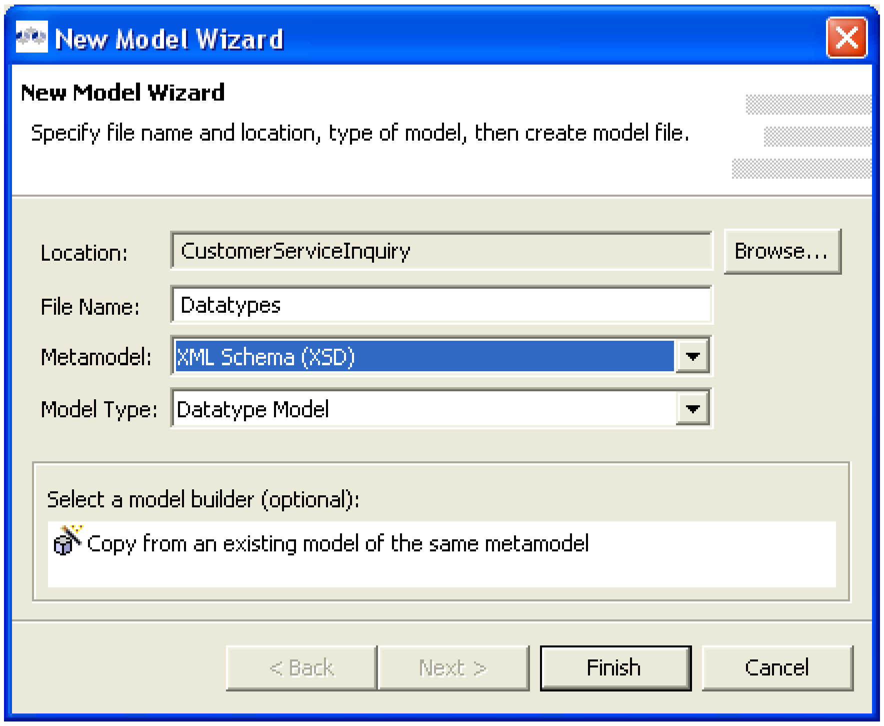
Each datatype model can contain one or more user-defined datatypes. A complete set of your organization's datatypes, contained within a metadata model, can be a complete data dictionary for your organization, providing users a single place to learn about your site's information types.

Creating the Datatype Model

You can create a datatype model the same way that you create physical or virtual models from other metamodels. Your datatype model, however, is a physical model to include in projects where you want to use the models.

To create a derived datatype model:

1. From the menu, select **New > Metadata Model**.
2. Click the **Next** button. Enter a **File Name**. Select the Metamodel type **XML Schema (XSD)**. The Model Type defaults to **Datatype Model**.


New Model Wizard

New Model Wizard

Specify file name and location, type of model, then create model file.

Location: CustomerServiceInquiry

File Name: Datatypes

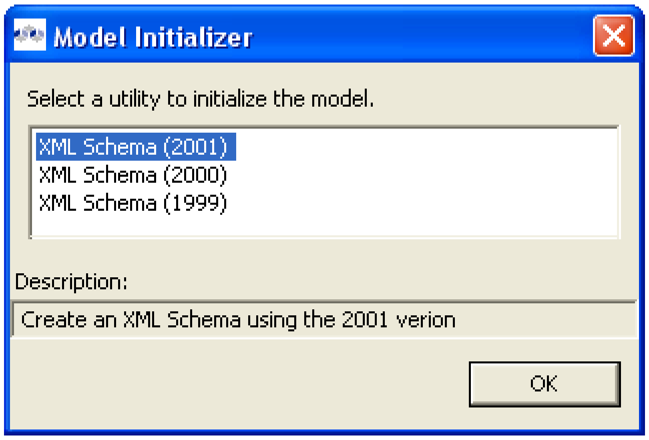
Metamodel: XML Schema (XSD)

Model Type: Datatype Model

Select a model builder (optional):

Copy from an existing model of the same metamodel

3. Click the **Finish** button.



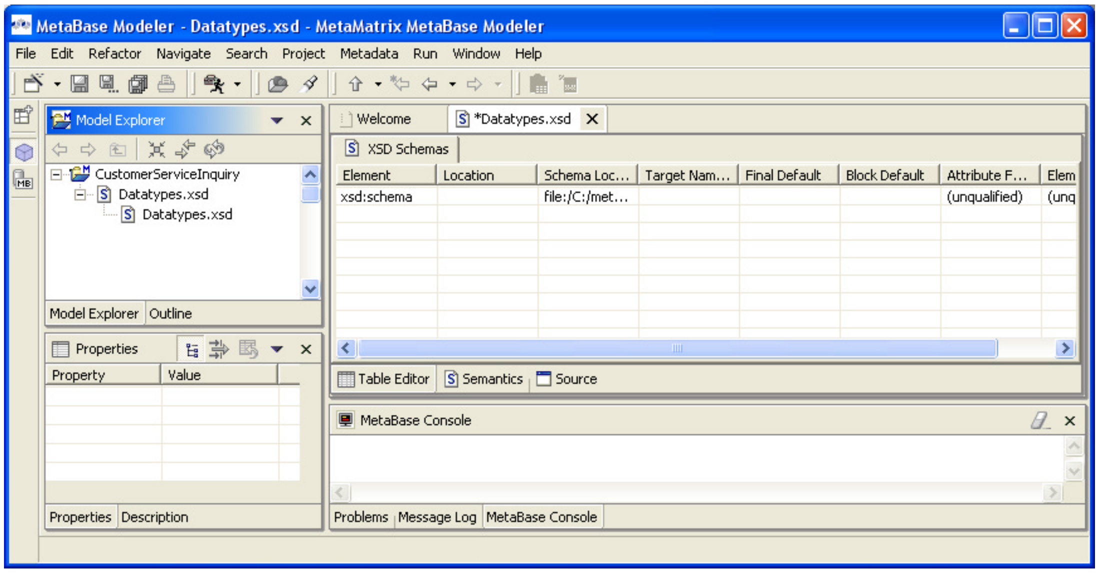
Select a utility to initialize the model.

XML Schema (2001)
XML Schema (2000)
XML Schema (1999)

Description:

Create an XML Schema using the 2001 verion

4. Next you are presented with a selection menu for the version of XML schema being applied.



MetaBase Modeler - Datatypes.xsd - MetaMatrix MetaBase Modeler
File Edit Refactor Navigate Search Project Metadata Run Window Help
Model Explorer
CustomerServiceInquiry
Datatypes.xsd
Datatypes.xsd
Model Explorer Outline
Properties
Property Value
Properties Description
Welcome
*Datatypes.xsd
XSD Schemas
Table Editor Semantics Source
MetaBase Console
Problems Message Log MetaBase Console

Element	Location	Schema Loc...	Target Nam...	Final Default	Block Default	Attribute F...	Elem
xsd:schema		file:/C:/met...				(unqualified)	(unq

5. By right-clicking on the schema diagram in the Editor Panel, you can select **New Child** and then add atomic, list, union datatypes, and a host of other definition types.

Chapter 9: Creating and Editing Meta Objects

CREATING META OBJECTS

When you model your existing data sources or create your transformed virtual classes, you need to create meta objects to represent the information.

You can create these meta objects in the following ways:

- Using the **Model Explorer view** to create the meta objects. This method organizes the meta objects in a hierarchical fashion, offering a quick way to start your metadata model. For more information, see "Creating Meta Objects on the Model Explorer View."
- Using the **Table Editor** to rapidly enter multiple meta objects.
- Using diagrams in the **Editor Panel** view.

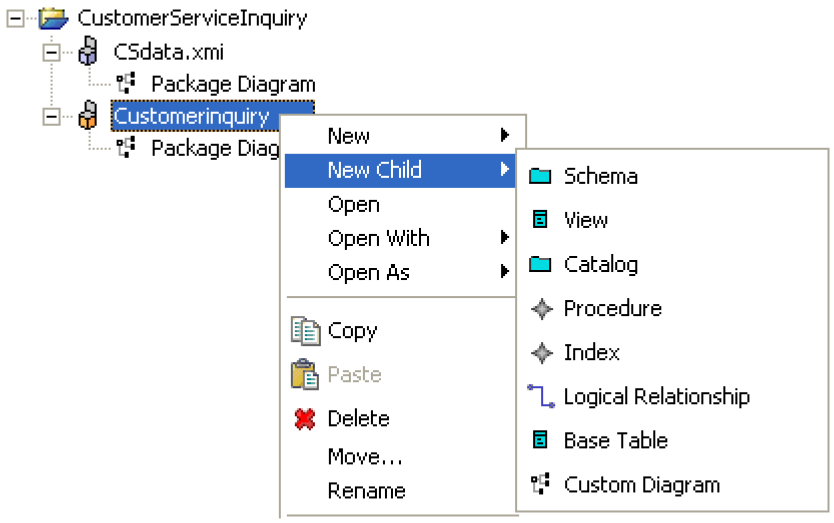
CREATING META OBJECTS ON THE MODEL EXPLORER VIEW

You can create meta objects directly on the **Model Explorer** view. The MetaBase Modeler constrains the types of meta objects you can create based upon the meta object you select and the metamodel of the metadata model in which you want to create the new meta object. You cannot create a Column attribute in a Stored Procedure class, nor can you create a Column meta object in a model based on the Data Access metamodel.

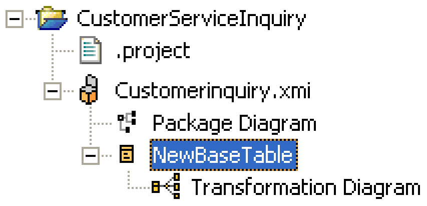
To create meta objects on the **Model Explorer** view:

1. Select the parent meta object to which you want to add a child. For example, you can add a package to a package or an attribute to a class.

2. Right-click on the meta object. From the pop-up menu, select **New Child**. You can now select the meta object you would like to add.



3. The new meta object displays on the **Model Explorer** view.



4. The new object is highlighted for renaming.

VIEWING META OBJECTS IN THE TABLE EDITOR

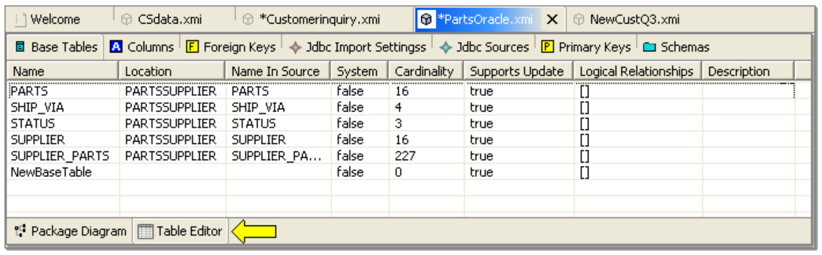
Using the Table Editor

The MetaBase Modeler includes a table-based meta object viewer and editor, called the **Table Editor**. Using the Table Editor, you can review meta objects, edit existing meta objects, and create new objects quickly in the **Editor Panel** view.

To open the Table Editor:

1. On the **Model Explorer** or **Outline** tab, select the model or meta objects you want to view or edit in the table.

2. Switch the **Editor Panel** view to the **Table Editor** view by clicking the tab at the bottom of the view.



Name	Location	Name In Source	System	Cardinality	Supports Update	Logical Relationships	Description
PARTS	PARTSSUPPLIER	PARTS	false	16	true	[]	
SHIP_VIA	PARTSSUPPLIER	SHIP_VIA	false	4	true	[]	
STATUS	PARTSSUPPLIER	STATUS	false	3	true	[]	
SUPPLIER	PARTSSUPPLIER	SUPPLIER	false	16	true	[]	
SUPPLIER_PARTS	PARTSSUPPLIER	SUPPLIER_PA...	false	227	true	[]	
NewBaseTable			false	0	true	[]	

The tabs that display depend upon the model's metamodel and the contents of each metamodel.

3. Once you have opened the Table Editor, you can:

   - Edit the existing properties. For more information, see "Editing Meta Objects in the Table Editor."

   - Add a new meta object. For more information, see "Creating Meta Objects in the Table Editor."

   - Add meta objects from your workspace to the table.

   - Paste meta object information from your clipboard into the table. For more information, see "Pasting into the Table Editor."

   - Print your tables.

Editing Meta Objects in the Table Editor

You can use the Table Editor to edit properties of a metaobject in the table.

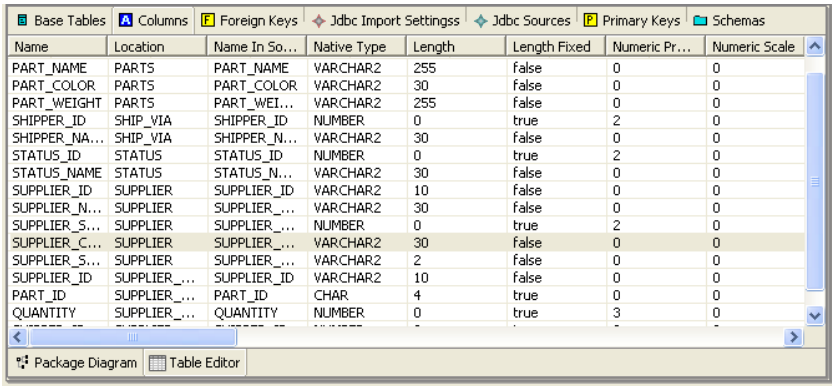
Typically, you can edit properties that use an edit box or a drop-down list to enter their values on the **Properties view**, such as **Datatype** and **Name In Source**. You cannot edit business metadata, such as keywords or descriptions.

To edit meta objects in a table:

1. On the **Model Explorer** view, select the meta objects you want to view or edit in the table.

2. From the **Editor Panel** view, select the tab to toggle on the **Table Editor**.

3. The **Table Editor** displays the contents of each metamodel. This sample displays a model using the Relational metamodel:



Name	Location	Name In So...	Native Type	Length	Length Fixed	Numeric Pr...	Numeric Scale
PART_NAME	PARTS	PART_NAME	VARCHAR2	255	false	0	0
PART_COLOR	PARTS	PART_COLOR	VARCHAR2	30	false	0	0
PART_WEIGHT	PARTS	PART_WEI...	VARCHAR2	255	false	0	0
SHIPPER_ID	SHIP_VIA	SHIPPER_ID	NUMBER	0	true	2	0
SHIPPER_NA...	SHIP_VIA	SHIPPER_N...	VARCHAR2	30	false	0	0
STATUS_ID	STATUS	STATUS_ID	NUMBER	0	true	2	0
STATUS_NAME	STATUS	STATUS_N...	VARCHAR2	30	false	0	0
SUPPLIER_ID	SUPPLIER	SUPPLIER_ID	VARCHAR2	10	false	0	0
SUPPLIER_N...	SUPPLIER	SUPPLIER_...	VARCHAR2	30	false	0	0
SUPPLIER_S...	SUPPLIER	SUPPLIER_...	NUMBER	0	true	2	0
SUPPLIER_C...	SUPPLIER	SUPPLIER_...	VARCHAR2	30	false	0	0
SUPPLIER_S...	SUPPLIER	SUPPLIER_...	VARCHAR2	2	false	0	0
SUPPLIER_ID	SUPPLIER_...	SUPPLIER_ID	VARCHAR2	10	false	0	0
PART_ID	SUPPLIER_...	PART_ID	CHAR	4	true	0	0
QUANTITY	SUPPLIER_...	QUANTITY	NUMBER	0	true	3	0

4. You can click certain properties and manually edit them in the **Table Editor**.

5. The changes you make impact your model.

Creating Meta Objects in the Table Editor

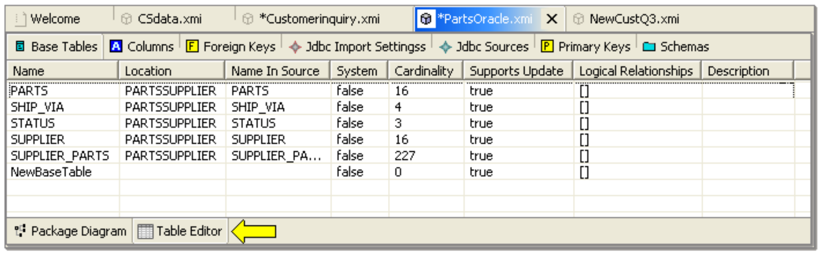
You can create meta objects in the Table View. When you create a meta object, you create it as a sibling of other meta objects in the table; it is the same type of meta object as the tab you are viewing and belongs to the same parent meta object.

For example, if you are reviewing columns in the table, you can add other columns, and they will belong to the same parent base table, view, or result set as the others in the table. If your table displays more than one set of columns, belonging to more than one parent class, you can select in which class you want to create the new column.

To create meta objects in a table:

1. On the **Model Explorer** view, select the meta objects you want to view in the table.

2. Switch the **Editor Panel** view to the **Table Editor** view by clicking the tab at the bottom of the view.



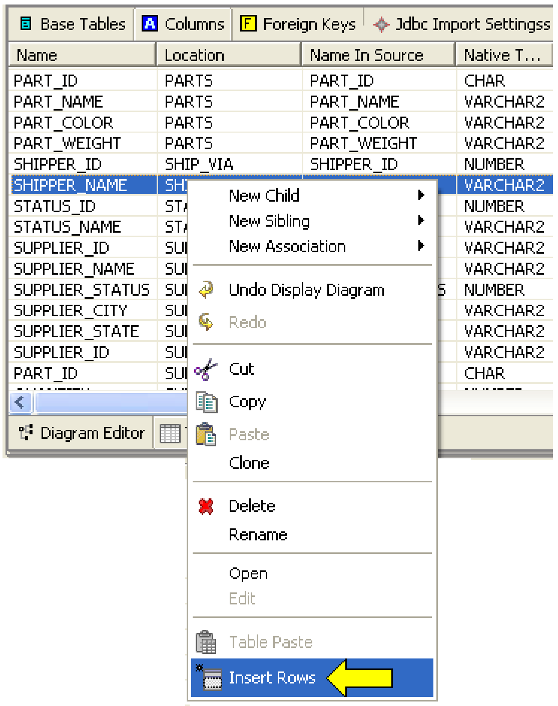
Name	Location	Name In Source	System	Cardinality	Supports Update	Logical Relationships	Description
PARTS	PARTSSUPPLIER	PARTS	false	16	true	[]	
SHIP_VIA	PARTSSUPPLIER	SHIP_VIA	false	4	true	[]	
STATUS	PARTSSUPPLIER	STATUS	false	3	true	[]	
SUPPLIER	PARTSSUPPLIER	SUPPLIER	false	16	true	[]	
SUPPLIER_PARTS	PARTSSUPPLIER	SUPPLIER_PA...	false	227	true	[]	
NewBaseTable			false	0	true	[]	

3. The Table Editor displays details for the meta object you selected.

4. Click the tab of the meta object type you want to add.

5. In the table, click a child of the parent meta object to which you want to add the new meta object.

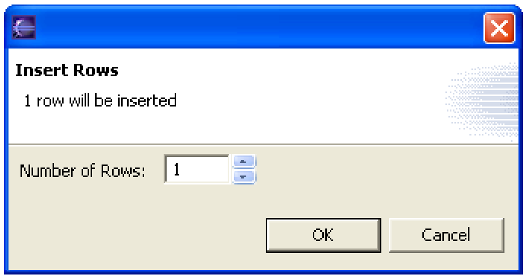
6. From the **Editor Panel** view, right-click on metadata in the Name column. Select **Insert Rows** from the drop-down menu.



Name	Location	Name In Source	Native T...
PART_ID	PARTS	PART_ID	CHAR
PART_NAME	PARTS	PART_NAME	VARCHAR2
PART_COLOR	PARTS	PART_COLOR	VARCHAR2
PART_WEIGHT	PARTS	PART_WEIGHT	VARCHAR2
SHIPPER_ID	SHIP_VIA	SHIPPER_ID	NUMBER
SHIPPER_NAME	SHI		VARCHAR2
STATUS_ID	STA		NUMBER
STATUS_NAME	STA		VARCHAR2
SUPPLIER_ID	SUI		VARCHAR2
SUPPLIER_NAME	SUI		VARCHAR2
SUPPLIER_STATUS	SUI	S	NUMBER
SUPPLIER_CITY	SUI		VARCHAR2
SUPPLIER_STATE	SUI		VARCHAR2
SUPPLIER_ID	SUI		VARCHAR2
PART_ID	SUI		CHAR

7. You also can use the **Insert Rows into Table** button in the top menu bar of the modeler.

8. In the **Insert Rows** dialog box, use the up and down arrow buttons to enter the number of rows you want to insert.



Insert Rows
1 row will be inserted

Number of Rows: 1

9. Click **OK**. A new row displays immediately beneath the row you clicked.

You can edit the properties of this new meta object. The new meta object becomes part of your model; you will see it in your workspace when you exit the Table Editor.

Pasting into the Table Editor

If you have a Microsoft Excel spreadsheet or other file that contains meta object property information, you can copy that information onto your operating system's clipboard and paste the information into the Table Editor.

The contents of this file must be tab-separated, but not comma-separated. The MetaBase Modeler will paste this information as metamodel-specific meta objects of a certain type, determined by the tab onto which you paste this information.

This paste function enables you to paste a block of contiguous rows and columns of property values into the table. This means you can populate many columns of information, not necessarily all. For example, if you have a spreadsheet containing meta object Names, Datatypes, and Descriptions, you can arrange your table and paste only that information into the table and can set other property values later. You cannot, however, paste namespace or location information into the table.

To paste items from the clipboard:

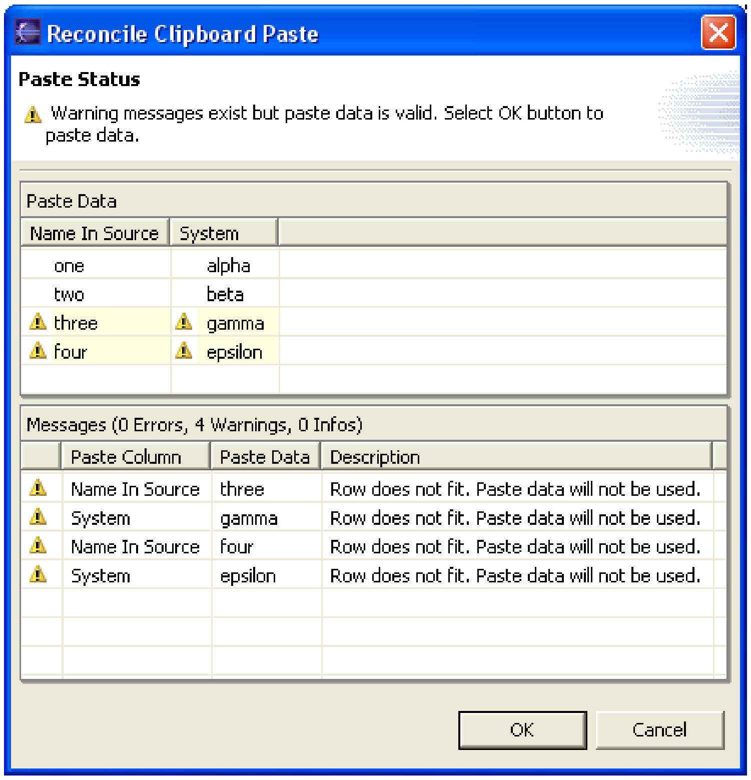
1. Toggle your **Editor Panel** to **Table Editor** using the tab at the bottom of the view.
2. Copy the information for the meta objects you want to create onto the system clipboard from Microsoft Excel.
3. Click the tab that corresponds to the meta object type this information represents. For example, to paste classes, click the **Base Tables**, **Views**, or **Groups** tab.
4. From the **Editor Panel** view, select **Insert Rows**. Repeat this step once for each row you want to paste; if you copied 20 rows to the clipboard, you must insert 20 rows.
5. Click the first of the new rows in the left-most column, typically **Name**.
6. From the **Editor Panel** view, select **Paste Clipboard Contents into Table**.

The rows you selected display in the model. These new meta objects become a part of your metadata model. You can create relationships, derive virtual metadata, and use these meta objects as any others.

You might encounter difficulties if you try to paste too many rows or if you try to paste invalid information.

Pasting Too Many Rows

Your paste operation might encounter problems if you attempt to paste more rows than you have created new rows. If you do, the **Reconcile Clipboard Paste** dialog box displays.


Reconcile Clipboard Paste

Paste Status

⚠ Warning messages exist but paste data is valid. Select OK button to paste data.

Paste Data

Name In Source	System
one	alpha
two	beta
⚠ three	⚠ gamma
⚠ four	⚠ epsilon

Messages (0 Errors, 4 Warnings, 0 Infos)

	Paste Column	Paste Data	Description
⚠	Name In Source	three	Row does not fit. Paste data will not be used.
⚠	System	gamma	Row does not fit. Paste data will not be used.
⚠	Name In Source	four	Row does not fit. Paste data will not be used.
⚠	System	epsilon	Row does not fit. Paste data will not be used.

You can click **OK** to paste the information anyway; this discards the information that is highlighted. Or you can click **Cancel** and alter either your existing table by adding rows for that information, or the dimensions of the data you are importing.

Other Limitations

There is no ability to re-order table columns once they have been imported. All modifications must be done in Microsoft Excel prior to the cut-and-paste.

The Table Editor feature is not designed for creating new objects. This means that you cannot "paste" an entirely new object, specifying its parent name.

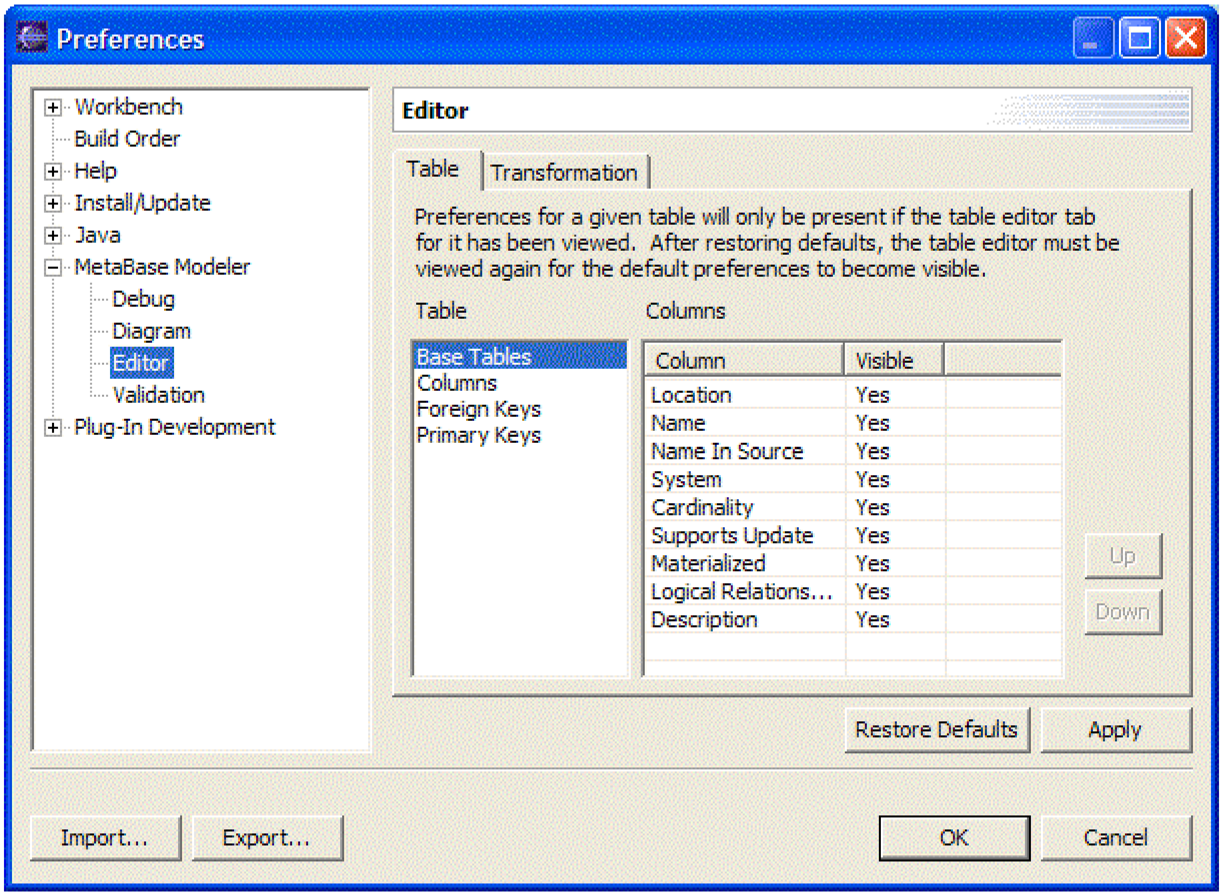
Location is not a valid paste field. You cannot change an object's parent in the table.

Since each tab is only one entity type (one for tables, one for columns), it is problematic to build an entire model by cutting and pasting spreadsheets.

Table Editor Column Sorting and Hiding

You can specify the order of columns in the table editor. You can also hide columns. Settings become available for a specific table type after it has been opened once. These settings can be accessed in one of two ways. First, you can navigate to **MetaBase Modeler Preferences > Editor** in the preferences dialog.

There will be a two tabs present, one for **Tables** and one for **Transformations** with the new settings on the **Tables** tab.



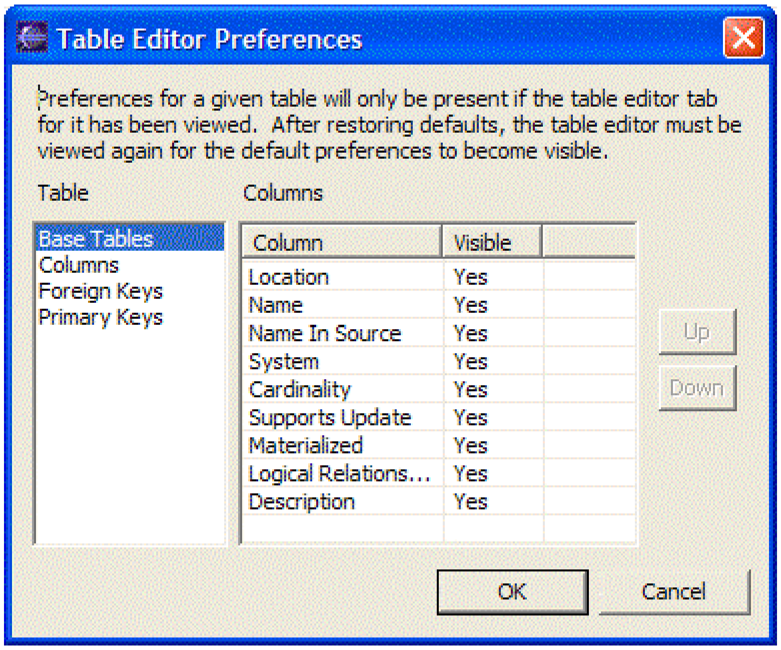
Preferences for a given table will only be present if the table editor tab for it has been viewed. After restoring defaults, the table editor must be viewed again for the default preferences to become visible.

Table: Base Tables, Columns, Foreign Keys, Primary Keys

Columns

Column	Visible
Location	Yes
Name	Yes
Name In Source	Yes
System	Yes
Cardinality	Yes
Supports Update	Yes
Materialized	Yes
Logical Relations…	Yes
Description	Yes

The settings can also be accessed by right-clicking on the table editor and selecting **Table Editor Preferences**. Table types with columns available for preference settings are shown in the left hand column.



Preferences for a given table will only be present if the table editor tab for it has been viewed. After restoring defaults, the table editor must be viewed again for the default preferences to become visible.

Table: Base Tables, Columns, Foreign Keys, Primary Keys

Columns

Column	Visible
Location	Yes
Name	Yes
Name In Source	Yes
System	Yes
Cardinality	Yes
Supports Update	Yes
Materialized	Yes
Logical Relations…	Yes
Description	Yes

Individual columns are shown for the selected table type in the right hand column. Clicking on the **yes** or **no** option next to the column name determines column visibility. Column order is changed by selecting a column and using the **up** and **down** buttons next to the column list to change its order. Selecting the **Restore Defults** button resets the default order and visibility of columns for a particular table. Note that there is no **Restore Defaults** button on the dialog presented with the right-click method. You can still restore defaults by selecting the table in the left hand column, right-clicking, and choosing the **Restore Defaults** menu option. Either method will cause the table to be removed from the preferences list. The table will not reappear until you have opened a new editor that contains that specific table type.

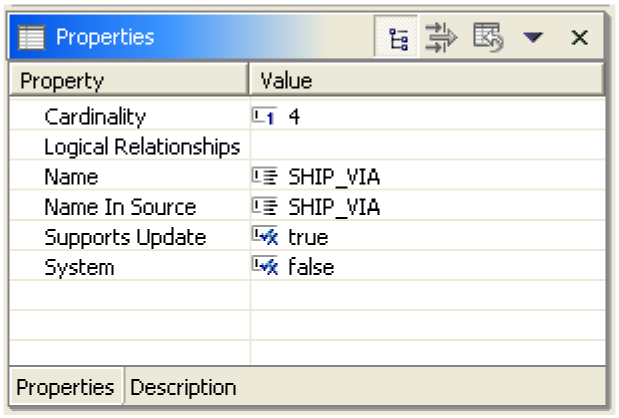
EDITING META OBJECT PROPERTIES

In addition to the Table Editor, you can use the **Properties** tab of the **Properties view** to change a meta object's properties. To edit a meta object's properties on the **Properties view**:

1. On the **Model Explorer** tab, select the meta object you want to modify.

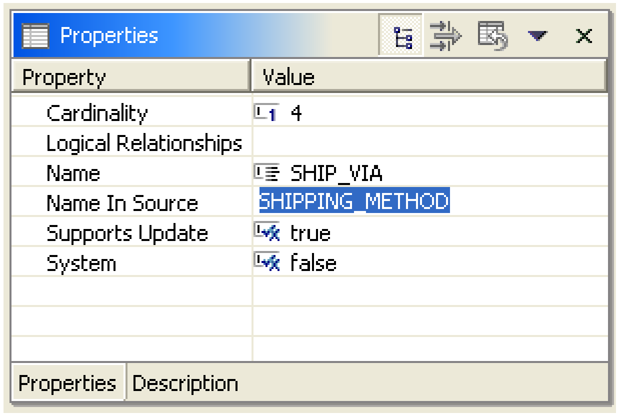
NOTE: *You can also select a meta object in the **Editor Panel view**.*

2. The properties display on the **Properties** view. The names and natures of the properties depend upon the type of meta object you selected.



Property	Value
Cardinality	4
Logical Relationships	
Name	SHIP_VIA
Name In Source	SHIP_VIA
Supports Update	true
System	false

3. Click the value beside the property you want to change and type the new value.

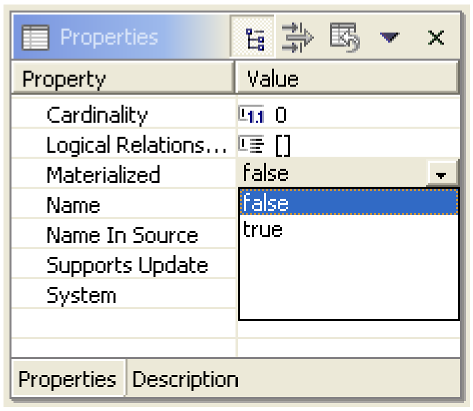


Property	Value
Cardinality	4
Logical Relationships	
Name	SHIP_VIA
Name In Source	SHIPPING_METHOD
Supports Update	true
System	false

4. Note that some properties are read-only; you cannot select or modify all properties. The MetaBase Modeler saves the changes you have made to your local directory. Remember, the changes you make do not reside in the MetaBase Repository until you add the model or check in your changes using the MetaBase Repository Manager.

Materialized Views

There is a new property on virtual tables named **Materialized**. Setting this property's value to **true** (the default is **false**) allows the data generated for this Virtual table to be stored as a materialized view using the MetaMatrix Server.



Property	Value
Cardinality	0
Logical Relations...	[]
Materialized	false
Name	false
Name In Source	true
Supports Update	
System	

Restore Default Values

There is a **Restore Default Value** button in the properties toolbar. Clicking this button (when active) will reset the highlighted property to its default null value.

MANIPULATING META OBJECTS

Once you create meta objects in one or more metadata models, the MetaBase Modeler provides you several handy ways to manipulate them. The MetaBase Modeler supports the common functions of cut, copy, and paste, but also adds a special clone function to

You can:

- Cut a meta object and its children from their current location and place it on the clipboard to paste elsewhere.
- Copy a meta object and its children, leaving it in the current location and placing a copy of it on the clipboard to paste elsewhere.
- Paste a cut or copied meta object and its children into a different location.
- Paste a cut or copied meta object and its children into a different location in a different metamodel. For more information, see "Pasting Meta Objects."
- Clone a meta object and its children, placing a copy of the meta object and its children in the same location, so that the copied meta object is a sibling to that which it copies.